1/5/1 (Item 1 from file: 351)
DIALOG(R)File 351:Derwent WPI

011738756 **Image available**
WPI Acc No: 1998-155666/ **199814**
XRPX Acc No: N98-124292

Multi-thread execution method using highly efficient microprocessor - involves providing instruction that decides order by which thread is generated and eliminated, to limit data dependence relationships between parent and element thread within stipulated thread-generating procedure

Patent Assignee: NEC CORP (NIDE)
Inventor: TORII S
Number of Countries: 002 Number of Patents: 003
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week
JP 10027108	A	19980127	JP 96183533	A	19960712	199814 B
US 6389446	B1	20020514	US 97888590	A	19970630	200239
US 20020147760	A1	20021010	US 97888590	A	19970630	200269
			US 2002114075	A	20020403	

Priority Applications (No Type Date): JP 96183533 A 19960712
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
JP 10027108	A		26	G06F-009/46	
US 6389446	B1			G06F-009/00	
US 20020147760	A1			G06F-009/00	Div ex application US 97888590 Div ex patent US 6389446

Abstract (Basic): JP 10027108 A

The method involves preparing several thread-executing portions that perform parallel threading operations. The parent thread which includes thread-generating and completing procedures is supplied into one of the executing portions. The portion containing the parent thread is made to execute a thread-generating procedure that forms an element thread containing another set of generating and completing procedures. One of the thread-executing portions then performs a completing procedure to terminate the parent thread, after which an element thread is supplied to one of the other executing portions from which another thread-generating procedure is to be executed.

One thread-generating procedure is stipulated to be performed for each executing portion. The element thread then provides an instruction that decides the order by which a thread is to be generated and eliminated. As a result, the data dependence relationship between the parent and element thread is limited only to the stipulated procedure per executing portion.

ADVANTAGE - Simplifies thread management to operation cost. Improves threading capability by restricting speculative thread-forming.

Dwg.1/24

Title Terms: MULTI; THREAD; EXECUTE; METHOD; HIGH; EFFICIENCY; MICROPROCESSOR; INSTRUCTION; DECIDE; ORDER; THREAD; GENERATE; ELIMINATE; LIMIT; DATA; DEPEND; RELATED; PARENT; ELEMENT; THREAD; STIPULATED; THREAD ; GENERATE; PROCEDURE
Derwent Class: T01
International Patent Class (Main): G06F-009/00; G06F-009/46
File Segment: EPI

1/5/2 (Item 1 from file: 347)
DIALOG(R)File 347:JAPIO

05744008 **Image available**
METHOD FOR THREAD EXECUTION

PUB. NO.: 10-027108 A]
PUBLISHED: January 27, 1998 (19980127)
INVENTOR(s): TORII ATSUSHI
APPLICANT(s): NEC CORP [000423] (A Japanese Company or Corporation), JP (Japan)
APPL. NO.: 08-183533 [JP 96183533]
FILED: July 12, 1996 (19960712)
INTL CLASS: [6] G06F-009/46; G06F-009/46
JAPIO CLASS: 45.1 (INFORMATION PROCESSING -- Arithmetic Sequence Units)
JAPIO KEYWORD: R131 (INFORMATION PROCESSING -- Microcomputers & Microprocessers)

ABSTRACT

PROBLEM TO BE SOLVED: To attain efficient thread scheduling by simplifying the thread scheduling of a multi-thread.

SOLUTION: One thread execution part executes a thread generating instruction 2 for a thread #0, generates a new thread #1 and executes a thread end instruction 3 for the thread #0 to end the execution of the thread 90. The other thread generating instruction 2 executes a thread generating instruction 2 for the thread #1, generates a new thread #2 and executes a thread end instruction 3 for the thread #1 to end the execution of the thread #1. Thus the generation frequency of a slave thread by a master thread is regulated only to once and the order of generation and end of threads is uniquely regulated. Since the dependence relation 4 between threads is regulated only to a direction from the master thread to the slave thread, the cost of thread scheduling can be reduced.

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－27108

(43)公開日　平成10年(1998)１月27日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/46	３４０		Ｇ０６Ｆ　9/46	３４０Ｂ	
	３６０			３６０Ｂ	

審査請求　有　　請求項の数10　ＯＬ　（全 26 頁）

(21)出願番号　特願平8－183533

(22)出願日　平成８年(1996)７月12日

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号

(72)発明者　鳥居　淳
東京都港区芝五丁目７番１号　日本電気株式会社内

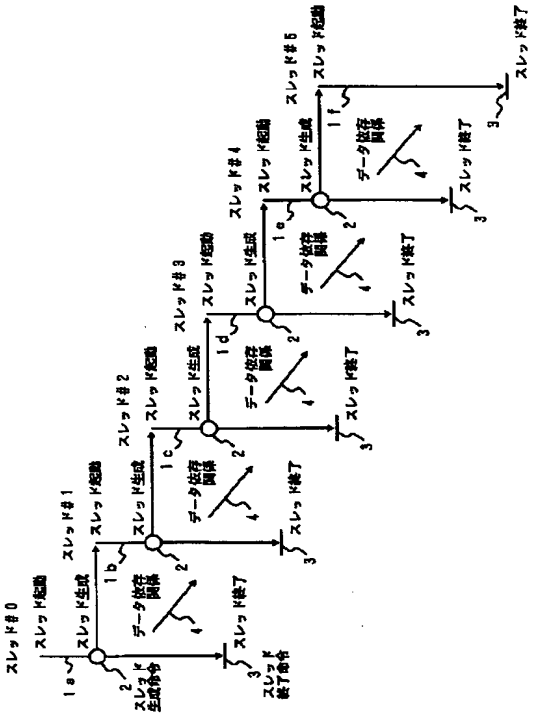
(74)代理人　弁理士　後藤　洋介　（外２名）

(54)【発明の名称】　スレッド実行方法

(57)【要約】

【課題】　マルチスレッドのスレッドスケジューリングを簡単化することによって、効率的なスレッドスケジューリングを実現する。

【解決手段】　一つのスレッド実行部は、スレッド＃０のスレッド生成命令２を実行して、新しいスレッド＃１を生成し、その後、スレッド＃０のスレッド終了命令３を実行して、スレッド＃０の実行を終了する。もう一つのスレッド実行部は、スレッド＃１のスレッド生成命令２を実行して、新しいスレッド＃２を生成し、その後、スレッド＃１のスレッド終了命令３を実行して、スレッド＃１の実行を終了する。このように親スレッドが子スレッドを生成する回数を高々１回と規定し、スレッド生成、終了順序を一意に規定する。このようなスレッド間の依存関係４を親スレッドから子スレッド方向のみに限定することによって、スレッドスケジューリングのコストを低減する。



1

【特許請求の範囲】

【請求項１】　複数のスレッドを並列に実行する複数のスレッド実行部を用意し、

前記複数のスレッド実行部の一つに、スレッド生成手続き及びスレッド終了手続きを含む親スレッドを供給し、

前記複数のスレッド実行部の前記一つに前記親スレッドの前記スレッド生成手続きを実行させて、スレッド生成手続き及びスレッド終了手続きを含む子スレッドを生成させ、その後、前記複数のスレッド実行部の前記一つに、前記親スレッドの前記スレッド終了手続きを実行させて、前記親スレッドを終了させ、

前記子スレッドを前記複数のスレッド実行部の異なる一つに供給し、前記複数のスレッド実行部の前記異なる一つに前記子スレッドを前記親スレッドとして実行させるスレッド実行方法であって、

前記複数のスレッド実行部の各々に前記スレッド生成手続きを実行させる回数を、前記親スレッド及び前記子スレッドの各々に対して高々１回に規定する規定と、前記子スレッドは前記親スレッドが生存中の間は終了できないという規定とを設けることによって、スレッドの生成順序及び消滅順序を一意に定め、

前記親スレッド及び前記子スレッド間のデータ依存関係を親スレッドから子スレッド方向へ限定することを特徴とするスレッド実行方法。

【請求項２】　前記複数のスレッド実行部に加え、スレッド実行部の状態を管理する共有のスレッド管理部を用意し、そのスレッド管理部において、スレッド実行部の状態とスレッドの親子関係情報を保持し、

子スレッドの生成および親スレッドの終了を行なう場合に、スレッド実行部から子スレッド生成処理要求およびスレッド終了処理要求をスレッド管理部に送信し、

スレッド管理部は要求を行なったスレッド実行部が、前記規定を満たすことを調べ、

前記規定が満たされている場合には、スレッド生成処理もしくは終了処理を要求したスレッド実行部に対して指示し、

スレッド管理部の持つ内部情報を更新することを特徴とする請求項１に記載のスレッド実行方法。

【請求項３】　前記複数のスレッド実行部の各々にスレッド管理部を設け、このスレッド管理部の間をリング状に結合する情報伝達線を設け、この情報伝達線によってスレッドの親子関係情報及びスレッドの実行状況の情報を隣接スレッドに伝達し、

前記複数のスレッド実行部の各々が子スレッドの生成を行なう際には、規定の方向の隣接スレッド実行部が、待機状態にある場合には、子スレッド生成を隣接スレッド実行部に依頼し、実行状態にある場合には、スレッド生成を待機状態になるまで遅延し、

前記複数のスレッド実行部の各々がスレッドの終了を行なう場合には、自スレッドに対する親スレッド終了を確認して後い、スレッドを終了することを特徴とする請求項１に記載のスレッド実行方法。

2

【請求項４】　スレッド実行部もしくはスレッド管理部に処理上の不都合が生じた場合には、情報伝達線をスレッド管理部を通さずに両隣接スレッド管理部に接続することによってシステムの冗長性を向上させたことを特徴とする請求項３に記載のスレッド実行方法。

【請求項５】　スレッド生成に必要なプログラムカウンタの値や親スレッド番号、親スレッドから継承する情報をスレッド情報として蓄えるスレッドバッファ部を一つ設け、

スレッドバッファ部はスレッド管理部及びスレッド実行部と、情報を交換する手段を具備し、

待機状態であるスレッド実行部がシステム中に存在しない場合には、スレッドバッファに対して必要な情報を送り、

スレッド実行部で実行していたスレッドが終了し、待機状態になった場合に、スレッドバッファ部にスレッド情報が存在する場合には、スレッドバッファ部に格納されていたスレッドを起動することを特徴とする請求項２～４のいづれかに記載のスレッド実行方法。

【請求項６】　親スレッドが子スレッドを生成する前に、親スレッド側の手続きによって子スレッドの実行上の属性を設定し、

その後生成された子スレッドは、その属性にしたがった動作を行ない、

スレッド生成後に親スレッドは別の手続きによって子スレッドの実行上の属性を変更することを特徴とする請求項１に記載のスレッド実行方法。

【請求項７】　子スレッド側で親スレッドの実行を待ち合わせる手続きを用意し、手続き実行後は親スレッドが規定の状態に達するまで、子スレッドの実行を中断することを特徴とする請求項１および請求項６のいづれかに記載のスレッド実行方法。

【請求項８】　親スレッドがプログラム実行上の正当性が確定する前に、子スレッド生成の投機的に生成するための特殊手続きと、正当性が確認された後にその結果を子スレッドに伝えるための特殊手続きと、不当な生成である場合に子スレッドの実行を取り消すための特殊手続きを加え、

また、スレッド実行部に、実行に伴う作用を取り消すための機構を加え、この機構の許容する範囲内でスレッドの仮実行を行ない、

親スレッドのスレッド実行部が子スレッドの実行が正当であるという特殊手続きを実行すると、その情報が子スレッドを実行しているスレッド実行部に伝えられ、子スレッド実行部は通常の実行状態に遷移し、

親スレッドのスレッド実行部がスレッド実行取消手続きを実行すると、その情報が子スレッドを実行しているスレッド実行部に伝えられ、子スレッドの実行を取り消す

機能を備え、
さらに、親スレッドのスレッド実行部がスレッド実行取消手続きを実行した場合には、スレッド生成を行なっていないという状態に戻すことを特徴とする請求項1、6、及び7のいづれかに記載のスレッド実行方法。
【請求項9】　請求項2に記載のスレッド実行方法において、親スレッドで指定した子スレッドの属性を保持しておく手段を備え、子スレッドの状態は本部の値の論理和をとることによって決定し、
このスレッド管理部を用いることによって、各スレッド実行部の実行状態を遷移させて、請求項6に記載のスレッド実行方法を実現することを特徴とするスレッド実行方法。
【請求項10】　請求項3に記載のスレッド実行方法において、親スレッドが設定する子スレッドの実行情報を、リング上に結合する情報伝達線で子スレッドを実行するスレッド実行部に伝達し、
親スレッドからの状態と次スレッドによる子スレッドの設定の論理和をさらに子スレッドに伝え、
それらの論理和によって自スレッドの状態を決定することを特徴とするスレッド実行方法。
【発明の詳細な説明】
【0001】
【発明の属する技術分野】本発明は、複数のスレッドを同時に実行する高性能マイクロプロセッサを用いたマルチスレッド実行方法に関する。
【0002】
【従来の技術】問題の持つ並列性を活用し、単一のプログラムを複数の命令流（スレッド）群に分割して、それらを並列に実行することによって、性能を向上させるプログラム実行方法として、マルチスレッド実行方法が数多く提案されている。このマルチスレッド実行方法によれば、スレッドはスレッド生成を意味するフォーク動作によって生成される。ここでは、フォーク動作を行なったスレッドを親スレッド、生成された新しいスレッドを子スレッドと呼ぶ。スレッドはマルチスレッド化されたプログラムにおいて、ある規定された動作を行なった後に消滅する。つまり、プログラム実行過程において、数々のスレッドの生成と終了が繰り返されることになる。
【0003】このスレッドは、プロセッシングユニットなどのスレッド実行部に割り当てられる。スレッド実行部が物理的に複数存在するシステムでは、複数のスレッドを同時に実行することが可能となり、逐次処理からの性能向上が期待できる。また、個々のスレッド実行部に複数のスレッドを割り当てることによって、同期ミスや、資源競合、キャッシュミスが生じた際に現在実行しているスレッドを待機状態にして、別のスレッドを起動することによって、これらの要因による遅延を見かけ上隠蔽し資源の利用効率をあげることが可能になる。
【0004】このようなマルチスレッド実行方法を説明

した代表的な文献としては、R．S．Nikhil等による“A Multithreaded Massively Parallel Architecture”（Proceedingsof the 19th Annual International Symposium on Computer Architecture，ページ156－167，May 1992）や、D．E．Culler等による“Finegrain Parallelism with Minimal Hardware Support：A Compiler－ControlledThreaded Abstract Machine”（Proceedings of the Fourth International Conference on Architectural Support forProgramming Languages and OperatingSystems，ページ164－175，April 1991）などがあげられる。
【0005】しかしながら、このスレッド生成、消滅、待機状態への変更、復帰などの発生頻度が他の処理に対して多い場合には、スレッド生成、消滅、待機状態への変更、復帰などのスレッドスケジューリングの効率によって並列処理全体の効率を決定することになり、このコストを低減することが、効率的な並列処理を行なう場合に肝要になる。プログラムの持つ並列性の少ない問題では、1スレッドあたりの処理量が小さくなったり、同時に存在するスレッド数が、スレッド実行部の数を大幅に上回る状態になると、スレッド実行コストやスレッドスケジューリングコストが飛躍的に増大するという欠点があった。
【0006】これに対して、処理量の小さな細粒度スレッドを効率的に実行するスレッド実行方法と実行装置が提案されている。例えば、細粒度スレッドの並列処理プロセッサの例として、Gurinder S．Sohi等が発表した論文“Multiscalar Processor”（The 22nd International Symposium on Computer Architecture，IEEE Computer Society Press，1995，ページ414－425）があげられる。これに開示されたマルチスカラー・プロセッサでは、単一のプログラムをいくつかの基本ブロックの集合である「タスク」に分割し、これを並列に実行処理できるプロセッサで処理する。
【0007】図23は、このマルチスカラー・プロセッサを例示している。マルチスカラー・プロセッサは、シーケンサ73、プロセッシングユニット74、結合ネットワーク75、データバンク76から構成される。プロセッシングユニット74は命令キャッシュ77、実行ユニット78、レジスタファイル79から構成され、シス

5

テムに複数存在する。また、プロセッシングユニット７４に対応してデータバンク７６も複数存在し、データバンク７６は、ＡＲＢ（アドレス・レゾリューション・バッファ：Ａｄｄｒｅｓｓ　Ｒｅｓｏｌｕｔｉｏｎ　Ｂｕｆｆｅｒ）８０、データキャッシュ８１とから構成される。複数のタスクの同時実行の管理は制御フロー情報が記載されたタスク記述子（ｔａｓｋ　ｄｅｓｃｒｉｐｔｏｒ）を用いてシーケンサ７３によって動的／静的に行なわれ、各プロセッシングユニット７４にタスクを割り付ける。シーケンサ７３は、タスクをプロセッシングユニット７４に割りつけると、次のタスクの割りつけを行なうべくタスク記述子を調べる。

【０００８】また、類似のスレッド実行方法としては、Ｐｒａｄｅｅｐ　Ｋ．Ｄｕｂｅｙ等発表した論文“Ｓｉｇｎｌｅ－Ｐｒｏｇｒａｍ　Ｓｐｅｃｕｌａｔｉｖｅ　Ｍｕｌｔｉｔｈｒｅａｄｉｎｇ（ＳＰＳＭ）Ａｒｃｈｉｔｅｃｔｉｒｅ：Ｃｏｍｐｉｌｅｒ－ａｓｓｉｓｔｅｄ　Ｆｉｎｅ－Ｇｒａｉｎｅｄ　Ｍｕｌｔｉｔｈｒｅａｄｉｎｇ”（Ｐｒａｒｌｌｅｌ　Ａｒｃｈｉｔｅｃｔｕｒｅｓ　ａｎｄ　Ｃｏｍｐｉｌａｔｉｏｎ　Ｔｅｃｈｎｉｑｕｅｓ，ＩＦＩＰ　１９９５）にも示されている。

【０００９】図２４はＳＰＳＭアーキテクチャにおけるスレッド実行方法の概念図である。図２４において、シングルプログラム８２中に、スレッド生成命令８４とスレッド待ち合わせ命令８５が組み込まれている。スレッド生成命令８４を実行すると、実行を先回りするフューチャースレッド８３を生成する。フューチャースレッド８３はスレッド待ち合わせ命令８５まで実行して、本来の実行が追いつくのを待ち合わせて結果をマージする。フューチャースレッド８３は本来の実行と並列に動作することによって、速度向上を図るというものである。

【００１０】

【発明が解決しようとする課題】しかしながら、前記のスレッド実行方法では、フォーク命令を実行したり、タスク記述子によってスレッドを起動する際に、待機状態のスレッド実行部が存在しない場合には、スレッドが生成できずプログラムが期待する振舞いができない現象が生じる可能性があった。また、スレッド生成をスレッド実行部が確保できるまで保留にした場合でも、その後のスレッド生成が継続されるため、スレッド数が増加するなどの欠点があった。また、スレッド間の対応関係を保持しておく必要や、スレッド管理は並列システム中で集中的に行なわなくてはならないので、スレッド管理部の制約によってスレッド実行部数が制限されてしまったり、スレッド管理方法自身が複雑になり、スレッド管理部のハードウェア化が簡単にはできないという欠点があった。

【００１１】本発明の課題は、スレッドスケジューリングコストを低減しつつ、前述の欠点を除去したスレッド実行方法を提供することにある。

6

【００１２】

【課題を解決するための手段】本発明によれば、複数のスレッドを並列に実行する複数のスレッド実行部を用意し、前記複数のスレッド実行部の一つに、スレッド生成手続き及びスレッド終了手続きを含む親スレッドを供給し、前記複数のスレッド実行部の前記一つに前記親スレッドの前記スレッド生成手続きを実行させて、スレッド生成手続き及びスレッド終了手続きを含む子スレッドを生成させ、その後、前記複数のスレッド実行部の前記一つに、前記親スレッドの前記スレッド終了手続きを実行させて、前記親スレッドを終了させ、前記子スレッドを前記複数のスレッド実行部の異なる一つに供給し、前記複数のスレッド実行部の前記異なる一つに前記子スレッドを前記親スレッドとして実行させるスレッド実行方法であって、前記複数のスレッド実行部の各々に前記スレッド生成手続きを実行させる回数を、前記親スレッド及び前記子スレッドの各々に対して高々１回に規定する規定と、前記子スレッドは前記親スレッドが生存中の間は終了できないという規定とを設けることによって、スレッドの生成順序及び消滅順序を一意に定め、前記親スレッド及び前記子スレッド間のデータ依存関係を親スレッドから子スレッド方向へ限定することを特徴とするスレッド実行方法が得られる。

【００１３】このように、本発明では、スレッド生成を１スレッドで高々１回と規定し、スレッドの生成順序及び終了順序を静的に一意に定める。これらスレッドの生成、終了は特殊命令を用いることによって行なう。また、スレッド間の依存関係を親スレッドから子スレッド方向に限定することにより、スレッド実行部が確保できない場合に実行が保留となる子スレッドの数は高々１スレッドに限定される。

【００１４】本スレッド実行方法に従えば、スレッド間の直接の関係は、一対一に限定されるため、スレッドのスケジューリング管理が容易になる。さらに、このスレッド実行方法に基づいたスレッド管理部を用意する。スレッド管理部は並列システム中に集中型と分散型が構築できる。分散型の場合は隣接するスレッド管理部とのみ交信を行なうことによってスレッド管理が実現できるため、スレッド実行部数が増えた場合も現実的に対応できる。

【００１５】

【発明の実施の形態】次に本発明の実施例について図面を参照して説明する。

【００１６】以下に述べる本発明の実施例のすべてについて特殊手続きを命令セットとして定義した特殊命令として示している。また、本発明は以下に述べる本発明の実施例に限定されるものではなく、当業者が種々の他の方法で本発明を容易に実施可能である。

【００１７】図１は、本発明の第１の実施例によるマルチスレッドの実行方法を説明するための図である。この

本発明の第１の実施例は請求項１の発明を実施するためのものである。

【００１８】図１において、スレッド生成命令２は、一つのスレッドから新しいスレッドを生成するためのものである。また、スレッド終了命令３は、当該スレッドの実行を終了するためのものである。

【００１９】例えば、スレッド生成命令２は、スレッド＃０（１ａ）からスレッド＃１（１ｂ）を生成する。スレッド終了命令３は、スレッド＃０（１ａ）の実行を終了する。別のスレッド生成命令２は、スレッド＃１（１ｂ）からスレッド＃２（１ｃ）を生成する。別のスレッド終了命令３は、スレッド＃１（１ｂ）の実行を終了する。更に別のスレッド生成命令２は、スレッド＃２（１ｃ）からスレッド＃３（１ｄ）を生成する。更に別のスレッド終了命令３は、スレッド＃２（１ｃ）の実行を終了する。同様に、スレッド＃３（１ｄ）からスレッド＃４（１ｅ）が生成された後、スレッド＃３（１ｄ）の実行は終了される。スレッド＃４（１ｅ）からスレッド＃５（１ｆ）が生成された後、スレッド＃４（１ｅ）の実行は終了される。

【００２０】このように、スレッド＃１（１ｂ）に対する親スレッドはスレッド＃０（１ａ）となり、スレッド＃１（１ｂ）に対する子スレッドはスレッド＃２（１ｃ）となる。スレッド間にデータ依存関係４がある場合、この依存は親スレッドから子スレッド方向に限定され、必要に応じて同期をとることによって、依存するデータの生産と消費のタイミングをとる。また、親スレッドのスレッド終了命令３が子スレッドのスレッド終了命令３より先に実行される必要があるので、子スレッドのスレッド終了命令３の実行は、親スレッドのスレッド終了命令３が実行されて親スレッドが終了されるまで、ウエイトされる。

【００２１】図２は、このようなマルチスレッドの実行方法に従って、４個のスレッド実行部＃０，＃１，＃２，及び＃３から構成される並列処理システム上で並列処理を行なった際のタイミングチャートを示す。図１及び図２を参照して、スレッド実行部＃０がスレッド＃０のスレッド生成命令２を実行すると、スレッド実行部＃０は新しいスレッド＃１を生成し、スレッド実行部＃１にスレッド＃１を供給する。スレッド実行部＃１はスレッド＃１のスレッド生成命令２を実行すると、スレッド＃２を生成し、スレッド実行部＃２にスレッド＃２を供給する。また、スレッド＃１はスレッド＃０が終了されるまで、終了されない。同様に、スレッド＃２はスレッド＃１が終了されるまで終了されない。

【００２２】このように、スレッド生成命令２の実行を、１スレッド中で高々１回に制限し、スレッドの生成順序と同様に終了させることによって、スレッド実行部＃０～＃３に割り付けられるスレッドが静的に確定できる。なお、あるスレッド実行部がスレッド生成命令２を

実行する際に、スレッドを実行していない他のスレッド実行部（すなわち、フリー状態であるスレッド実行部）が存在しない場合には、フリー状態であるスレッド実行部が生じるまで、前述のスレッド実行部はスレッド生成命令の実行をウェイトすることになる。

【００２３】このマルチスレッド実行方法に従ったスレッドの生成例についての例を述べる。以下のようなループ処理について考える。このコードは米国ＭＩＰＳ　Ｔｅｃｈｎｏｌｏｇｙ　ＩＮＣ．社のＲＩＳＣプロセッサＲ３０００シリーズの命令セットから簡単化のために分岐遅延スロットを取り除いたものである。このループでは、レジスタｒ１が０になるまでループをして、０ｘ４０ｆ０４４番地に示されるサブルーチンをコールするものである。

【００２４】

```
40f040 :    addu    r4,r14,r18
40f044 :    jal     0x40e99c <0x40e99c>
40f048 :    addiu   r17,r17,0x8
40f04c :    slti    r1,r17,0x40
40f050 :    sll     r14,r17,1
40f054 :    bne     r1,r0,-0x18 <0x40f040>
```

このループの１反復を１スレッドとして定義して、マルチスレッド化したコード例を以下に示す。

【００２５】

```
40f040 :    fork    0x10 <0x40f050>
40f044 :    addu    r4,R14,R18
40f048 :    jal     0x40e99c <0x40e99c>
40f04c :    term
40f050 :    addiu   r17,r17,0x8
40f054 :    slti    r1,r17,0x40
40f058 :    sll     r14,r17,1
40f05c :    bne     r1,r0,-0x20 <0x40f040>
```

この例ではスレッド生成命令をｆｏｒｋ命令、スレッド終了命令をｔｅｒｍ命令と定義しており、サブルーチン内によるメモリの書き替えアドレスは競合せず、また、サブルーチン内ではスレッド生成を行なっていないことを前提としている。

【００２６】本発明によって、スレッド間のスケジューリングを容易に静的に割り付けるこが可能となり、また同時に存在するスレッド数がスレッド実行部を上回ることがなくなるため、常にスレッド数の上限が保証され、効率的なスレッド管理が可能となる。

【００２７】図３は、本発明の第２の実施例によるマルチスレッドの実行方法を実施するための実行装置のブロック図である。この本発明の第２の実施例は請求項２の発明を実施するためのものである。

【００２８】図３の実行装置は、４スレッド並列実行型プロセッサである。図３において、スレッド実行部＃０（６ａ）、＃１（６ｂ）、＃２（６ｃ）、及び＃３（６ｄ）はスレッド管理部５と接続され、スレッド実行部６

（添字削除）間はスレッド実行部間交信バス７と接続されている。

【００２９】図４はスレッド管理部５の詳細なブロック図である。スレッド管理部５はスレッド管理シーケンサ８とスレッド状態テーブル９から構成されており、スレッド状態テーブル９ではスレッド実行部６の個数分のエントリが用意されている。各エントリはスレッド状態エントリ１０、親スレッド実行部番号エントリ１１、子スレッド実行部番号エントリ１２から構成される。スレッド状態エントリ１０は対応するスレッド実行部６のビジー状態／フリー状態を識別するために用いられ、親スレッド実行部番号エントリ１１及び子スレッド実行部番号エントリ１２は該当スレッド実行部６で実行されているスレッドの親スレッド及び子スレッドが実行されているスレッド実行部６の番号を示す。

【００３０】以下、スレッド生成時およびスレッド終了時のスレッド管理ユニット５の動作アルゴリズムを説明する。これらのアルゴリズムは主としてスレッド管理シーケンサ８の状態遷移で行なわれる。

【００３１】図５はスレッド管理シーケンサ８のスレッド生成時のアルゴリズムを示したフローチャートである。

【００３２】図３～図５を参照して、スレッド実行部６が、スレッド生成命令２を実行すると、スレッド管理部５に対して子スレッド生成要求を行なう。スレッド管理部５が子スレッド生成要求を受け付けると、そのスレッド実行部６で実行しているスレッドが、既に子スレッドを生成したか否かを調べる。これは、子スレッド実行部エントリ番号１２の内容を調べることによって行なわれる。ここで、スレッドを既に生成していれば、スレッド生成を行うことなく、エラーを表すエラー信号をスレッド実行部６に返信する。スレッドを生成していない場合には、次に、フリー状態のスレッド実行部６が存在するか否かを調べる。これは、スレッド状態エントリ１０の内容を調べることによって行なわれる。ここで、フリー状態のスレッド実行部６が存在しない場合には、現時点でのスレッド生成は物理的に不可能となるので、スレッド生成不可能を表す情報を、スレッド生成を要求したスレッド実行部６に返す。このスレッド実行部６は、他のスレッド実行部６がフリー状態になるまで待たされるか、ＯＳなどのシステムソフトウェアによってメモリなどに、子スレッド起動のための情報を退避するなどの処理が必要である。

【００３３】もし、待たされる場合には、他のスレッド実行部６がフリー状態になることは、スレッド実行部６から一定間隔で問い合わせたり、スレッド管理部５が、以前要求のあったスレッド実行部６の番号を記憶しておき、他のスレッド実行部６がスレッド終了した時に、要求したスレッド実行部６に対してその情報を伝えるなどの方法によって行なうことが可能である。また、メモリ

上に子スレッド起動情報を退避した場合は、フリー状態のスレッド実行部６が生じた際に、その情報からシステムソフトウェアによって新たに子スレッドを生成することになる。

【００３４】さて、フリー状態のスレッド実行部６が存在する場合には、まず、どのスレッド実行部６に子スレッドを実行させるか決定する。次に、スレッド管理テーブル９を更新する。まず、新たに子スレッドを実行するスレッド実行部６のスレッド状態エントリ１０をフリー状態からビジー状態に変更し、親スレッド実行部番号エントリ１１をスレッド生成要求を行なったスレッド実行部６の番号に設定する。次に、このスレッド生成要求を行なったスレッド実行部６の子スレッド実行部番号エントリ１２を設定する。次に、スレッド生成要求を行なったスレッド実行部６に子スレッド実行部６の番号を伝える。この情報を受けとったスレッド実行部６は、子スレッド実行部６に対してスレッド実行部間交信バス７を介して、子スレッドに引き継がれるデータやスレッドの開始アドレスなどを送信し、これによって子スレッドが実行されることになる。

【００３５】図６は、スレッドを終了する時のスレッド管理シーケンサ８の動作アルゴリズムのフローチャートを示す。

【００３６】図３、図４、及び図６を参照して、スレッドの終了は、スレッドの生成順序と同じ順序であることを保証する必要があるので、親スレッドが終了していない限り、終了することはできない。親スレッドの終了は、スレッド状態テーブル９の親スレッド実行部番号エントリ１１が、クリアされているか否かで判断し、クリアされていない場合には、スレッド実行部６を待ち状態にさせる。クリアされている場合には、スレッド状態テーブル９の要求スレッド実行部６のスレッド状態テーブル１０をフリー状態にし、子スレッド実行部番号エントリ１２に示されるエントリの親スレッド実行部番号エントリ１１をクリアする。その後、要求スレッド実行部６に対して、スレッド終了受理信号を返す。スレッド実行部６はこの受理信号を受けとった時点でスレッドを終了させる。

【００３７】このように、本発明ではマルチスレッドの管理をソフトウェアによってではなくすべてハードウェア論理によって実現可能のため、スレッド管理オーバヘッドが大幅に低減され、効率的なマルチスレッドの実行が可能になる。スレッド実行部６の数を上回るスレッドが生成された場合に、システムソフトウェアによってメモリにスレッド起動情報を退避する場合でも、退避されるスレッドは、上述したように１スレッドでフォーク１回と規定しているため、１スレッド分のみのメモリ領域だけ確保すれば十分である。

【００３８】なお、図３に示される並列処理システムにおいて、単一のタスクをマルチスレッドに分割して実行

するだけであれば、スレッド状態テーブル９の親スレッド実行部番号エントリ１１と子スレッド実行部番号エントリ１２を省略して、もっとも古い親スレッドを示すエントリを追加することでも同様の機能をより簡単に実現できるが、本実施例の場合は、全く依存関係のない複数のタスクをスレッド実行部６で空間分割して、例えば、スレッド実行部＃０（１６ａ）及び＃１（１６ｂ）は第１のタスク用、スレッド実行部＃２（６ｃ）及び＃３（１６ｄ）は第２のタスク用としてマルチタスク・マルチスレッドで実行することも可能となる。

【００３９】図７は、本発明の第３の実施例によるマルチスレッドの実行方法を実施するための実行装置のブロック図である。この本発明の第３の実施例は請求項３の発明を実施するためのものである。

【００４０】図７の実行装置は４スレッド並列実行型プロセッサである。図７において、スレッド実行部＃０（１４ａ）、＃１（１４ｂ）、＃２（１４ｃ）、及び＃３（１４ｄ）はそれぞれスレッド管理部＃０（１３ａ）、＃１（１３ｂ）、＃２（１３ｃ）、及び＃３（１３ｄ）と接続されており、隣接するスレッド実行部１４（添字削除）はスレッド実行部間交信バス１５によって単一方向リング状に接続されている。また、隣接するスレッド管理部１３（添字削除）は、スレッド実行部間交信バス１５と同一の単一方向で親スレッド情報伝達線１６によって接続されると共に、スレッド実行部間交信バス１５と逆の単一方向の子スレッド情報伝達線１７によって接続されている。このように各スレッド実行部１４やスレッド管理部１３はリング状に接続されているため、生成された新スレッドの供給はスレッド実行部間交信バス１５の方向に隣接するスレッド実行部１４のみに対してのみ可能である。

【００４１】図８は、スレッド管理部＃１（１３ｂ）の詳細を示す。以下にスレッド管理部＃１（１３ｂ）の構造及び動作を説明するが、他のスレッド管理部＃０（１３ａ）、＃２（１３ｃ）、及び＃３（１３ｄ）も構造及び動作において同様である。スレッド管理部１３ｂは、スレッド管理シーケンサ１８とスレッド状態テーブル１９とから構成されている。スレッド状態テーブル１９はスレッド状態エントリ２０、親スレッド識別エントリ２１、親スレッド終了判定論理部２２から構成される。

【００４２】親スレッド情報伝達線１６はシステム中に存在する最も古い親スレッド（最古親スレッド）が他のスレッド実行部１４に存在するか否かを示す信号線であり、スレッド管理部＃２（１３ｃ）への出力は、親スレッド識別エントリ２１と親スレッド終了判定論理部２２からの直接出力の論理和によって決定される。

【００４３】図９は親スレッド終了判定論理部２２の動作を説明するためのタイミングチャートである。図９において、（Ａ）はスレッド管理部＃０（１３ａ）からの親スレッド情報伝達線１６で、スレッド管理部＃１（１３ｂ）に対して、他のスレッド実行部１４に最古親スレッドが存在するか否かの信号を表す。（Ｂ）はスレッド管理シーケンサ１８からのスレッド実行部＃１（１４ｂ）へのスレッド終了通知信号を表す。（Ｃ）は論理和回路への直接出力信号を表す。（Ｄ）はスレッド状態テーブルの親スレッド識別エントリ２１への出力を表す。（Ｅ）は論理和回路の出力を表し、この値がスレッド管理部＃２（１３ｃ）への親スレッド情報伝達線１６の値となる。

【００４４】初期状態は、最古親スレッドはスレッド実行部＃０（１４ａ）で実行されているとする。この時には、（Ａ）及び（Ｅ）から、親スレッドが他に存在する（つまり親スレッドがスレッド実行部＃１（１４ｂ）及び＃２（１４ｃ）ではない）ということがスレッド実行部＃１（１４ｂ）及び＃２（１４ｃ）でわかる。サイクル１において最古親スレッドに対する実行をスレッド実行部＃０（１４ａ）が終了すると、サイクル２において、スレッド管理部＃０（１３ａ）は、（Ａ）に示すように、スレッド管理部＃１（１３ｂ）に対して、スレッド実行部＃１（１４ｂ）が実行しているスレッドより古い親スレッドは存在しないと通知する。これを受けて、同時に、（Ｄ）に示すように、親スレッド識別エントリ２１を親スレッドであるという状態を書き替える。また、１サイクル遅延のサイクル３において、（Ｃ）に示すように、論理和回路への直接出力をリセットする。これによって、スレッド管理部＃２（１３ｃ）によっては、（Ｅ）で示される親スレッド状態伝達線１６の変化は生じない。

【００４５】次に、スレッド実行部＃１（１４ｂ）で実行されていたスレッドが終了する場合について説明する。図９では、サイクル５で（Ｂ）に示すスレッド終了通知信号がその情報を伝える。すると、次のサイクル６で（Ｄ）に示す親スレッド識別エントリ２１を親スレッドではないという状態に書き替える。すると、（Ｅ）に示す親スレッド状態伝達線１６は、（Ｃ）及び（Ｄ）に示す論理和なので、スレッド実行部＃１（１４ｂ）より前には最古親スレッドが存在しないという情報となってスレッド管理部＃２（１３ｃ）に伝えられる。これらによって、分散するスレッド管理部１３に対してスレッド管理上必要な情報を提供する。

【００４６】図１０は図８のスレッド管理シーケンサ１８のスレッド生成時の動作アルゴリズムを示し、図１１は図８のスレッド管理シーケンサ１８のスレッド終了時の動作アルゴリズムを示す。

【００４７】図１０及び図１１の基本的なアルゴリズムは図５及び図６の場合と同じである。子スレッドの存在、及び子スレッド生成方向の隣接スレッドが最古の親スレッドであるかの情報は子スレッド情報伝達線１７（図７及び図８）によって、スレッド管理シーケンサ１８に伝えられる。

【００４８】図８、図９、及び図１０を参照して、スレッド管理部＃１（１３ｂ）のスレッド管理シーケンサ１８は、スレッド実行部＃１（１４ｂ）からの子スレッド生成要求を受けると、まず子スレッド情報伝達線１７の情報によって既に子にスレッドが生成されているか否かを調べる。ここで、スレッド管理部＃１（１３ｂ）のスレッド管理シーケンサ１８は、スレッドを既に生成していれば、スレッド生成を行なうことになく、エラーを表すエラー信号をスレッド実行部１４ｂに返信する。スレッド管理部＃１（１３ｂ）のスレッド管理シーケンサ１８は、スレッドを生成していない場合には、隣接スレッド実行部＃２（１４ｃ）が最古親スレッドを実行していないかを調べる。

【００４９】ここで、スレッド管理部＃１（１３ｂ）のスレッド管理シーケンサ１８は、スレッド実行部＃２（１４ｃ）が最古親スレッドを実行している場合には、現時点でのスレッド生成は物理的に不可能となるので、図５の場合と同様に、スレッド実行部＃１（１４ｂ）に、スレッド生成が可能となるまでウェイトさせたり、或いはシステムソフトウェアによってメモリに子スレッド起動情報を退避させるために、スレッド生成不可能を表す情報をスレッド実行部＃１（１４ｂ）に返し、後ほどスレッド実行部＃２（１４ｃ）が２スレッド終了した場合に、スレッド実行部＃１（１４ａ）に対してその旨の情報を伝える。

【００５０】スレッド管理部＃１（１３ｂ）のスレッド管理シーケンサ１８は、スレッド実行部＃２（１４ｃ）がフリー状態であれば、スレッド管理部＃１（１３ｂ）のスレッド状態エントリ２０をフリー状態からビジー状態に変更し、スレッド実行部＃１（１４ｂ）に要求受理信号を返す。この情報を受けとったスレッド生成部＃１（１４ｂ）は、スレッド実行部＃２（１４ｃ）に対してスレッド実行部間交信バス１５を介して、子スレッドに引き継がれるデータやスレッドの開始アドレスなどを送信し、これによって子スレッドが起動されることになる。

【００５１】図８、図９、及び図１１を参照して、スレッド実行部＃１（１４ｂ）のスレッド終了時のスレッド管理部＃１（１３ｂ）のスレッド管理シーケンサ１８の動作を説明する。スレッドの終了は、スレッドの生成順序と同じ順序であることを保証する必要があるので、親スレッドが終了していない限り、終了することはできない。親スレッドの終了は、親スレッド情報伝達線１６によって伝達される。親スレッドが終了していない場合には、スレッド終了できるまでスレッド実行部＃１（１４ｂ）を待ち状態にさせる。親スレッドが終了している場合（つまりスレッド実行部＃１（１４ｂ）が実行しているスレッドが最古親スレッドである場合）には、図６と同様の手順で処理を行ない、スレッド実行部＃１（１４ｂ）にスレッド終了許可を表すスレッド終了受理信号を送信する。スレッド実行部＃１（１４ｂ）はこれを受けとった時点でスレッドを終了させる。

【００５２】なお、スレッド管理部＃１（１３ｂ）のスレッド管理シーケンサ１８は、隣接親スレッドが終了すると、スレッド状態テーブル（スレッド管理テーブル）１９を更新し、親スレッド終了判定の出力をＮｏに遷移させるべく、親スレッド識別エントリ２１をＮｏに変更する。

【００５３】本発明によって、並列システム全体で共有するスレッド管理機構を用いずに、第１の実施例によるマルチスレッド実行方法が実現され、スレッド実行部の数を増やした場合にも、スレッド管理部が容易に構成可能である。

【００５４】図１２は、本発明の第４の実施例によるマルチスレッド実行方法を実施するための実行装置のブロック図である。この本発明の第４の実施例によるマルチスレッド実行方法は請求項４の発明を実施するためのものであり、並列処理システムに冗長性持たせ、耐故障性を持たせたものである。

【００５５】図１２の実行装置は、図７の実行装置と同様に、スレッド管理部２３ａ～２３ｄ、スレッド実行部２４ａ～２４ｄ、スレッド実行部間交信バス２５、親スレッド情報伝達線２６、及び子スレッド情報伝達線２７を有する。ここで、スレッド実行部間交信バス２５は各スレッド実行部２４（添字削除）をバイパスする拡張バスを有する。親スレッド情報伝達線２６も、各スレッド管理部２３（添字削除）をバイパスする拡張伝達線を有し、子スレッド情報伝達線２７も、各スレッド管理部２３をバイパスする拡張伝達線を有する。

【００５６】もし、スレッド実行部２４やスレッド管理部２３に故障等の不都合が生じた際には、問題のスレッド実行部２４やスレッド管理部２３を拡張バスや拡張伝達線を用いてバイパスすることによって、スレッド実行部２４は減じるものの、同一プログラムの実行が可能である。従って、本発明によって対故障性が必要な場合は並列処理システムにおいても、第１の実施例によるマルチスレッド実行方法が実現できる。

【００５７】図１３は、本発明の第５の実施例によるマルチスレッド実行方法を実施するための実行装置のブロック図である。この本発明の第５の実施例によるマルチスレッド実行方法は請求項５の発明を実施するためのものである。

【００５８】図１３の実行装置は、図３の実行装置と同様に、スレッド管理部２８と、スレッド実行部＃０（２９ａ）～＃３（２９ｄ）と、スレッド実行部間交信バス３０とを有する。スレッド管理部２８は、スレッド実行部＃０（２９ａ）～＃３（２９ｄ）と接続されている。スレッド実行部＃０（２９ａ）～＃３（２９ｄ）はスレッド実行部間交信バス３０に共通に接続されている。

【００５９】この実行装置は、更に、スレッド管理部２

15

８とスレッド実行部＃０（２９ａ）～＃３（２９ｄ）とに接続されたスレッド情報退避バッファ３１を有する。このスレッド情報退避バッファ３１は、一つのスレッド実行部２９（添字削除）が、残りのすべてのスレッド実行部２９でスレッドを実行中の時に、最新子スレッドの生成命令を実行した場合、この子スレッド起動のために必要になるスレッド開始アドレス情報やデータを蓄えておくためのものである。

【００６０】図１４の実行装置は、図７の実行装置に対して、同様のスレッド情報退避バッファ３７を追加したものである。この図１４の実行装置は、、図７の実行装置と同様に、スレッド管理部＃０（３２ａ）～＃３（３２ｄ）と、スレッド実行部＃０（３３ａ）～＃３（３３ｄ）と、スレッド実行部間交信バス３４と、親スレッド情報伝達線３５と、子スレッド情報伝達線３６とを有する。スレッド情報退避バッファ３７は、一つのスレッド実行部３３（添字削除）が、残りのすべてのスレッド実行部３３でスレッドを実行中の時に、最新子スレッドの生成命令を実行した場合、この子スレッド起動のために必要になるスレッド開始アドレス情報やデータを蓄えておくためのものである。

【００６１】図１５は図１３及び図１４の実行装置の動作を説明するためのタイミングチャートである。図１５に示した動作は図２に示した動作と基本的には類似している。図１５には、（Ｅ）で示されるスレッド情報退避バッファの状態が追加されている。（Ｄ）で示されるスレッド実行部＃３がスレッド＃３を実行しており、スレッド＃３のスレッド生成命令を実行した時には、（Ａ）、（Ｂ）、及び（Ｃ）で示されるスレッド実行部＃０、＃１、及び＃２ではそれぞれスレッド＃０、＃１、及び＃２を実行中のため、フリー状態のスレッド実行部が存在しない。したがって、スレッド管理部２８（或いは３２）はスレッド退避バッファ３１（或いは３７）に、子スレッド起動情報を記憶させる。（Ｄ）のスレッド実行部＃３はそのままスレッド＃３の実行を継続する。

【００６２】（Ａ）で示されるスレッド実行部＃０がスレッド＃０の実行を終了すると、フリー状態になる。この時に、スレッド管理部２８（或いは３２）は、スレッド退避バッファ３１（或いは３７）から子スレッド起動情報を取り出し、（Ａ）で示されるスレッド実行部＃０にロードして、子スレッド、すなわちスレッド＃４を起動する。

【００６３】このように、本発明によってスレッド数が並列システムが有するスレッド実行部の数を上回る場合についても、完全にハードウェアによってスレッド管理が可能になる。第１の実施例による実行方法に従うことにより、スレッド退避バッファ３１（或いは３７）は、１スレッド分のスレッド起動情報を蓄える容量のみでよいため、現実的なスレッド管理が可能である。

16

【００６４】図１６は、本発明の第６の実施例によるマルチスレッド実行方法を説明するための図である。この本発明の第６の実施例によるマルチスレッド実行方法は請求項６の発明を実施するためのものであり、子スレッドの動作に関する属性を親スレッドの手続きによって指定するものである。

【００６５】図１６では属性指定の手続きを特殊命令によって行なう例を示している。スレッド＃０（３８ａ）は、スレッド生成命令３９によって、子スレッド＃１（３８ｂ）を生成する。この子スレッド＃１（３８ｂ）を生成するスレッド生成命令３９を実行する前に、子スレッドの属性を属性指定命令４１によって例えばＡに指定することによって、子スレッド＃１（３８ｂ）はこの属性Ａにしたがった動作を行なうことになる。属性の例としては、

・特定メモリアドレスへの参照制限
・特定資源へのアクセスの優先順位の設定
・特定命令の実行制限

などがあげられる。

【００６６】この属性の解除も親スレッド＃０が属性解除命令４２を実行することによって行なわれる、なお、この属性は子スレッドがさらにフォークした時に引き継がれる／引き継がれないという指定も親スレッド＃０から行なう。また、子スレッド＃１からも自分の状態を得る部分（命令や特殊レジスタの参照）を用意することも可能である。なお図１６において、４０はスレッド終了命令である。

【００６７】本発明によって、子スレッドの実行を制限したスレッド生成が可能になり、スレッドの記述性が向上し、問題の持っている並列をより利用できるようになる。

【００６８】図１７は、本発明の第７の実施例によるマルチスレッド実行方法を説明するための図である。この本発明の第７の実施例によるマルチスレッド実行方法は請求項７の発明を実施するためのものであり、親スレッドの実行状況を待たずに、子スレッドが実行を進めることを場合によって抑止するための発明である。

【００６９】例えば、二次元配列の演算を行なう場合に行方向の処理と列方向の処理を順に行なう必要が生じることがある。この際には、行方向の処理が終った時点ですべてのスレッドの同期をとった上で列方向の処理を開始する必要がある。この場合、第１の実施例で示したマルチスレッドの実行方法の場合は、子スレッド側からは親スレッドすなわちループの本体の演算部分の終了を検出できない。そこで、本発明では親スレッドの終了を待ち合わせる手続きを追加する。図１７においては、待ち合わせの手続きを、親スレッド待ち合わせ命令４６という特殊命令によって行なう例を示している。なお、図１７において、４３はスレッド、４４はスレッド生成命令、４５はスレッド終了命令である。

【００７０】図１７において、スレッド＃３において、親スレッド待ち合わせ命令４６を実行すると、スレッド＃２が終了するまで、スレッド＃３は待ち合わせ状態となる。このことにより、ループの並列処理などにおいて同期をとることが可能になる。例えば、第１の実施例の説明の際に例示したプログラムにおいては、以下に示すように、０ｘ４０ｆ０６０行にｐｗａｉｔという特殊命令を挿入することによって、このことを実現をしている。

【００７１】

```
40f040 :    fork    0x10 <0x40f050>
40f044 :    addu    r4,r14,r18
40f048 :    jal     0x40e99c <0x40e99c>
40f04c :    term
40f050 :    addiu   r17,r17,0x8
40f054 :    slti    r1,r17,0x40
40f058 :    sll     r14,r17,1
40f05c :    bne     r1,r0,-0x20 <0x40f040>
40f060 :    pwait
```

本実施例では親スレッドの終了を待ち合わせたが、その他に親スレッドの数が命令で定めた数以下になるまで待ち合わせをする手続きや、親スレッドの実行があるアドレスに到達するまで待ち合わせる手続きを用意することも本発明の範囲内である。

【００７２】図１８及び図１９は、本発明の第８の実施例によるマルチスレッド実行方法を説明するための図である。この本発明の第８の実施例によるマルチスレッド実行方法は請求項８の発明を実施するためのものであり、スレッド生成時機をより早めることによって、より多くの並列性を見い出すことを目的としている。即ち、この方法は、親スレッドの処理がある程度確定するまで子スレッドを確実に生成することがわからない場合には、投機的に子スレットを生成して、その後の親スレッドの処理の進み方によって、仮に生成した子スレッドを確定させたり、取り消しさせたりする方法である。

【００７３】図１８は投機が成功する場合を、図１９は投機が失敗する場合を示している。図１８において、スレッド＃０（４７ａ）が、投機的スレッド生成命令４９を実行すると、スレッド＃１（４７ｂ）が生成され、仮実行状態でスレッド＃１（４７ｂ）の実行が開始され、スレッド＃０はスレッド仮生成状態となる。スレッド＃１（４７ｂ）の仮実行状態でスレッド生成命令４８が実行されると、それによって生成されるスレッド＃２（４７ｃ）も、スレッド＃１（４７ｂ）の仮実行属性を引き継ぎ、仮実行状態で実行される。スレッド＃０（４７ａ）が条件分岐命令５３を実行後、この条件分岐によってスレッド生成の投機が正しいと確定すると、投機成功通知命令５１を実行する。これによって、スレッド＃１（４７ｂ）、スレッド＃２（４７ｃ）の仮実行状態が解除され、スレッド＃０（４７ａ）はスレッド生成状態となる。

【００７４】図１９においても同様に実行されるが、条件分岐命令５３が逆方向に分岐し、これによってスレッド生成の投機が不正であると確定する。この時、スレッド＃０（４７ａ）は投機失敗通知命令５２を実行する。これによって、スレッド＃１（４７ｂ）及びスレッド＃２（４７ｃ）の実行は取り消される。この投機失敗通知命令５２を実行することにより、スレッド＃０（４７ａ）はスレッド仮生成状態からスレッド未生成状態に戻り、再びスレッド生成命令４８を実行することが可能になる。

【００７５】なお図１８及び図１９において、５０はスレッド終了命令である。

【００７６】図１９のように投機的なスレッド生成に失敗した時には、投機的な実行による副作用が並列システムで生じないようにする必要がある。そのために共有メモリなどへの書き込みは抑制する必要がある。従って、図２０に示すようにスレッド実行部５４の中に、プロセッシングユニット５５、キャッシュメモリ５７の他に、仮実行待機中はキャッシュメモリ５７や、共有メモリ５８の更新を抑止するための仮実行用バッファ装置５６が必要となる。

【００７７】このようなマルチスレッドの実行方法は、例えば、以下のような誤差が収束するまで反復するようなループの演算を行なった際などに有効となる。

【００７８】

```
while （誤差が収束していない） {
    演算；
    誤差計算；
}
```

この場合、演算を行なった時点で子スレッドを投機的に生成し、次の演算を行なわせつつ、誤差計算結果が収束条件を満たしている場合には、次の演算を実行しているスレッドを取り消すということが、本マルチスレッド実行方法で実現可能である。

【００７９】図２１は本発明の第９の実施例によるマルチスレッド実行方法を説明するための図である。この本発明の第９の実施例によるマルチスレッド実行方法は請求項９の発明を実施するためのものであり、上述した第６の実施例で示したマルチスレッドの実行方法を実現すべく、上述した第２の実施例のスレッド管理部（図４）を拡張したものである。第９の実施例では、引き継ぐ属性を上述した第８の実施例に示した投機属性として、説明する。

【００８０】図２１は、スレッド管理部内のスレッド状態テーブル５９のブロック図である。このスレッド状態テーブル５９は、図４のスレッド状態テーブル５と同様に、スレッド状態エントリ６０と、親スレッド実行部番号エントリ６１と、子スレッド実行部番号エントリ６２とを有する。このスレッド状態テーブル５９は、更に、

投機的なスレッド生成に対処するために子スレッド実行状態エントリ６３を有する。子スレッド実行状態エントリ６３は、親スレッドから見て子スレッドの状態をどのように設定しているかを示している。したがって、図２１の例では、スレッド実行部＃０の子スレッド実行状態エントリ６３が投機状態にセットされているので、スレッド実行部＃１、スレッド実行部＃２で実行されているスレッドは仮実行状態で実行される。これは、図１８に示すようにスレッド＃０が投機的スレッド生成命令４９を実行し、スレッド＃１が通常のスレッド生成命令４８を実行したことを示している。

【００８１】この時、スレッド実行部＃２の実行状態はスレッド実行部＃０の子スレッド実行状態エントリ６３とスレッド実行部＃１の子スレッド実行状態エントリ６３の論理和によって投機状態となる。スレット実行部＃０が投機成功通知命令５１を実行した時点で、スレッド実行部＃０の子スレッド実行状態エントリ６３は確定実行状態となり、スレッド実行部＃１及びスレッド実行部＃２で実行しているスレッドは両者とも仮実行状態から確定実行状態に遷移する。

【００８２】このように、本発明によってスレッド状態テーブル５９によって集中的にスレッド管理を行ない、子スレッド以下は親スレッドの設定状態の論理和をとることによって、子スレッド以下の状態を容易に決定することが可能になる。

【００８３】図２２は本発明の第１０の実施例によるマルチスレッド実行方法を説明するための図である。この本発明の第１０の実施例によるマルチスレッド実行方法は請求項１０の発明を実施するためのものであり、上述した第６の実施例で示したマルチスレッドの実行方法を実現すべく、上述した第３の実施例のスレッド管理部（図７及び図８）を拡張したものである。第１０の実施例では、引き継ぐ属性を上述した第８の実施例に示した投機属性として、説明する。

【００８４】図２２は、スレッド管理部＃１のブロック図である。このスレッド管理部＃１は、図８のスレッド管理部＃１と同様に、スレッド管理シーケンサ６４と、スレッド状態テーブル６５と、親スレッド情報伝達線７０と、子スレッド情報伝達線７１とを有する。スレッド状態テーブル６５は、図８のスレッド状態テーブル１９と同様に、スレッド状態エントリ６６と、親スレッド識別エントリ６７と、親スレッド終了判定論理部６８とを有する。

【００８５】図２２のスレッド管理部＃１は、更に、子スレッド状態設定情報伝達線７２を有する。この子スレッド状態設定情報伝達線７２は、親スレッド情報伝達線７０と同一の方向の単方向の情報伝達線である。また、図２２のスレッド管理部＃１のスレッド状態テーブル６５は、更に、子スレッド実行状態エントリ６９を有する。

【００８６】スレッド管理部＃１において実行されるスレッドの状態はスレッド管理部＃０からの子スレッド状態設定情報伝達線７２の入力によって決定される。また、スレッド管理部＃２への子スレッド状態設定情報伝達線７２の出力は、自分が最古親スレッドであることが、親スレッド識別エントリ６７で示されていない場合には、スレッド管理部＃０からの子スレッド状態設定情報伝達線７２の入力と子スレッド実行状態エントリ６９の値の論理和によって生成される。自分が最古親スレッドであることが、親スレッド識別エントリ６７で示されている場合には、子スレッド実行状態エントリ６９の値がそのまま出力される。

【００８７】図１８及び図１９で説明した投機的スレッド生成命令４９で、子スレッドを生成した場合には、そのスレッド管理部の子スレッド実行状態エントリ６９を投機状態にセットする。このことによって、本スレッド以降に生成されるスレッドはすべて仮実行状態になり、本スレッドが投機成功命令５１を実行するまでこの仮状態が続くことになる。

【００８８】本発明によって、第３の実施例の利点を活かしたまま、第８の実施例で示した投機的なスレッド生成が実現可能となる。

【００８９】なお、上述した第９の実施例及び第１０の実施例において、スレッド実行部にはスレッド実行を取り消す機能が必要である。また、投機失敗通知命令５２（図１９）によってスレッド実行が取り消された場合には、スレッド状態エントリ６０（図２１）及び６６（図２２）をフォーク後状態からフォーク前状態に戻す必要がある。また、本実施例では、第８の実施例の属性を継承することについて示したが、他の親スレッド属性を子スレッドに引く継ぐことも同様に可能である。

【００９０】

【発明の効果】以上説明したように本発明によれば、スレッド生成、終了を順序つけ、スレッド生成を高々１回に規定することによって、スレッド管理の大幅な簡略化が可能となり、現実的なハードウェア規模でスレッド管理部のハードウェア化が可能となった。この結果、スレッド管理のコストが大幅に減少し、細粒度スレッドによる並列化に対しても大きな性能向上が期待できる。また、スレッドの生成を投機的に行なうことにより、問題の持つ並列性が限られる場合にも、性能向上が可能となった。

【図面の簡単な説明】

【図１】本発明の第１の実施例によるマルチスレッドの実行方法を説明するための図である。

【図２】図１のマルチスレッドの実行方法の動作を説明するためのタイミングチャートである。

【図３】本発明の第２の実施例によるマルチスレッドの実行方法を実施するための実行装置のブロック図である。

21

【図４】図３の実行装置のスレッド管理部のブロック図である。

【図５】図４のスレッド管理部のスレッド管理シーケンサのスレッド生成時の動作を説明するためのフローチャートである。

【図６】図４のスレッド管理部のスレッド管理シーケンサのスレッド終了時の動作を説明するためのフローチャートである。

【図７】本発明の第３の実施例によるマルチスレッドの実行方法を実施するための実行装置のブロック図である。

【図８】図７の実行装置のスレッド管理部のブロック図である。

【図９】図８のスレッド管理部の親スレッド終了判定論理部の動作を説明するためのタイミングチャートである。

【図１０】図８のスレッド管理シーケンサのスレッド生成時の動作を説明するためのフローチャートである。

【図１１】図８のスレッド管理シーケンサのスレッド終了時の動作を説明するためのフローチャートである。

【図１２】本発明の第４の実施例によるマルチスレッドの実行方法を実施するための実行装置のブロック図である。

【図１３】本発明の第５の実施例によるマルチスレッドの実行方法を実施するための実行装置のブロック図である。

【図１４】本発明の第５の実施例によるマルチスレッドの実行方法を実施するためのもう一つの実行装置のブロック図である。

【図１５】図１３及び図１４の実行装置の動作を説明するためのタイミングチャートである。

【図１６】本発明の第６の実施例によるマルチスレッドの実行方法を説明するための図である。

【図１７】本発明の第７の実施例によるマルチスレッドの実行方法を説明するための図である。

【図１８】本発明の第８の実施例によるマルチスレッドの実行方法の投機成功時の動作を説明するための図である。

【図１９】本発明の第８の実施例によるマルチスレッドの実行方法の投機失敗時の動作を説明するための図である。

【図２０】本発明の第８の実施例によるマルチスレッドの実行方法を実行する実行装置のスレッド実行部のブロック図である。

【図２１】本発明の第９の実施例によるマルチスレッドの実行方法を実行する実行装置ののスレッド状態テーブルのブロック図である。

【図２２】本発明の第１０の実施例によるマルチスレッドの実行方法を実行する実行装置のスレッド管理部のブロック図である。

22

【図２３】従来のマルチスカラー・プロセッサのブロック図である。

【図２４】従来のＳＰＳＭアーキテクチャにおけるスレッド実行方法を説明するための図である。

【符号の説明】

２　スレッド生成命令
３　スレッド終了命令
４　データ依存関係
５　スレッド管理部
６（添字削除）　スレッド実行部
７　スレッド実行部間交信バス
８　スレッド管理シーケンサ
９　スレッド状態テーブル
１０　スレッド状態エントリ
１１　親スレッド実行部番号エントリ
１２　子スレッド実行部番号エントリ
１３（添字削除）　スレッド管理部
１４（添字削除）　スレッド実行部
１５　スレッド実行部間交信バス
１６　親スレッド情報伝達線
１７　子スレッド情報伝達線
１８　スレッド管理シーケンサ
１９　スレッド状態テーブル
２０　スレッド状態エントリ
２１　親スレッド識別エントリ
２２　親スレッド終了判定論理部
２３（添字削除）　スレッド管理部
２４（添字削除）　スレッド実行部
２５　スレッド実行部間交信バス
２６　親スレッド情報伝達線
２７　子スレッド情報伝達線
２８　スレッド管理部
２９（添字削除）　スレッド実行部
３０　スレッド実行部間交信バス
３１　スレッド情報退避バッファ
３２（添字削除）　スレッド管理部
３３（添字削除）　スレッド実行部
３４　スレッド実行部間交信バス
３５　親スレッド情報伝達線
３６　子スレッド情報伝達線
３７　スレッド情報退避バッファ
３９　スレッド生成命令
４０　スレッド終了命令
４１　属性指定命令
４２　属性解除命令
４４　スレッド生成命令
４５　スレッド終了命令
４６　親スレッド待ち合わせ命令
４８　スレッド生成命令
４９　投機的スレッド生成命令

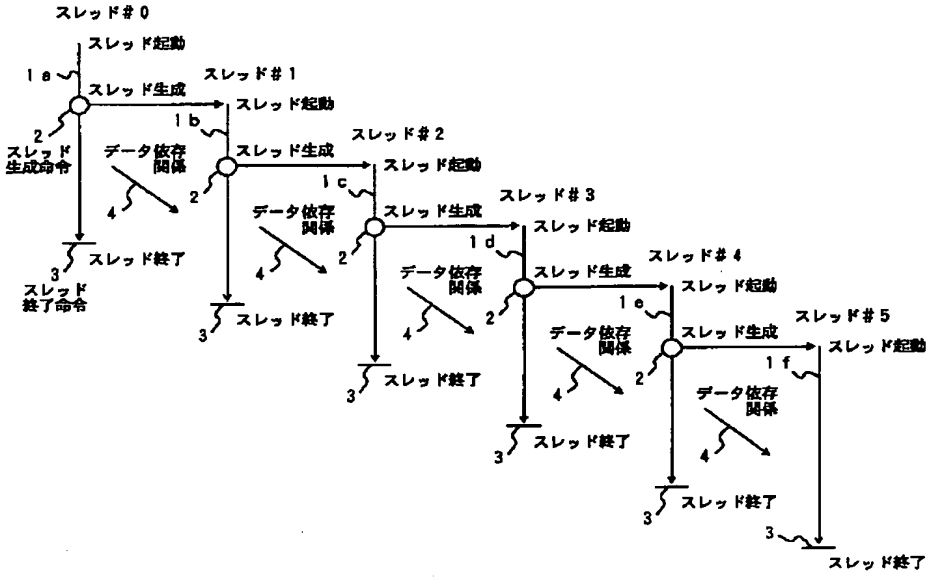
23

５０　スレッド終了命令
５１　投機成功通知命令
５２　投機失敗通知命令
５３　条件分岐命令
５９　スレッド状態テーブル
６０　スレッド状態エントリ
６１　親スレッド実行部番号エントリ
６２　子スレッド実行部番号エントリ
６３　子スレッド実行状態エントリ

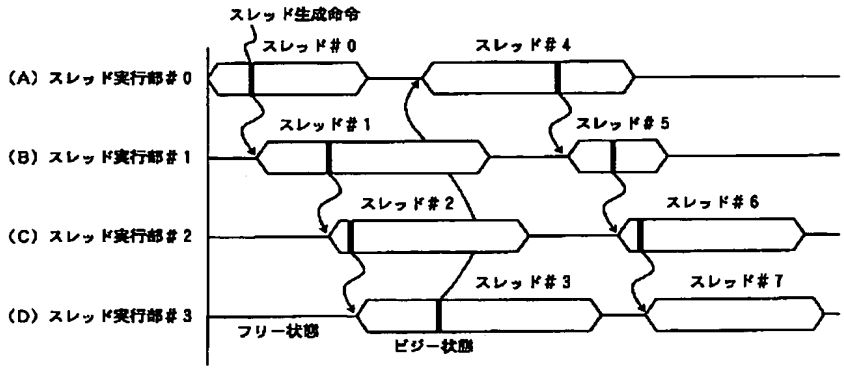
24

６４　スレッド管理シーケンサ
６５　スレッド状態テーブル
６６　スレッド状態エントリ
６７　親スレッド識別エントリ
６８　親スレッド終了判定論理部
６９　子スレッド実行状態エントリ
７０　親スレッド情報伝達線
７１　子スレッド情報伝達線
７２　子スレッド状態設定情報伝達線

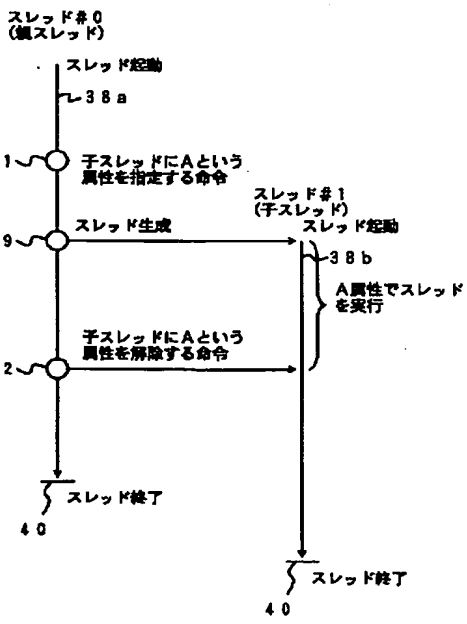
【図１】



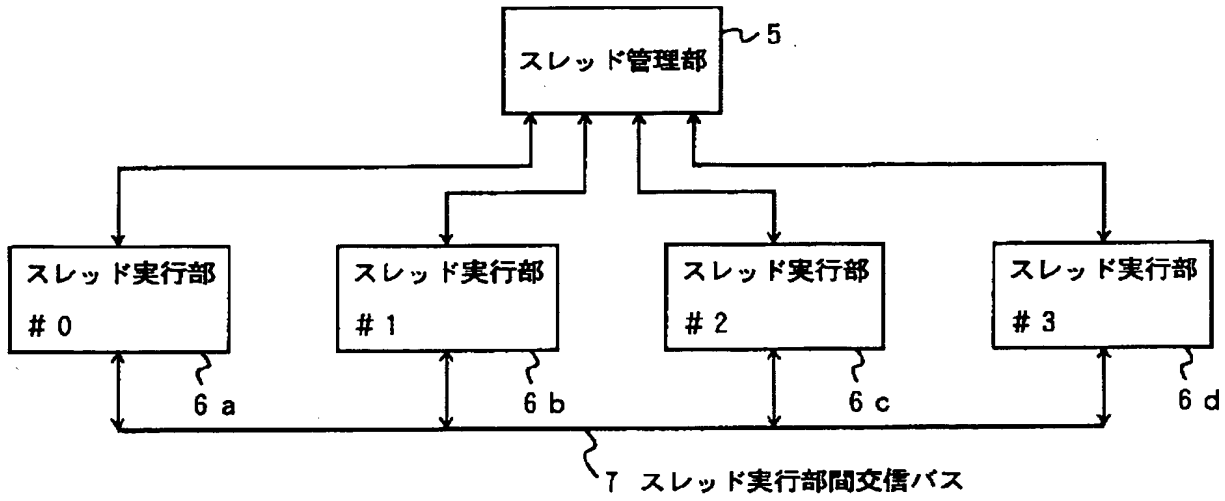
【図２】



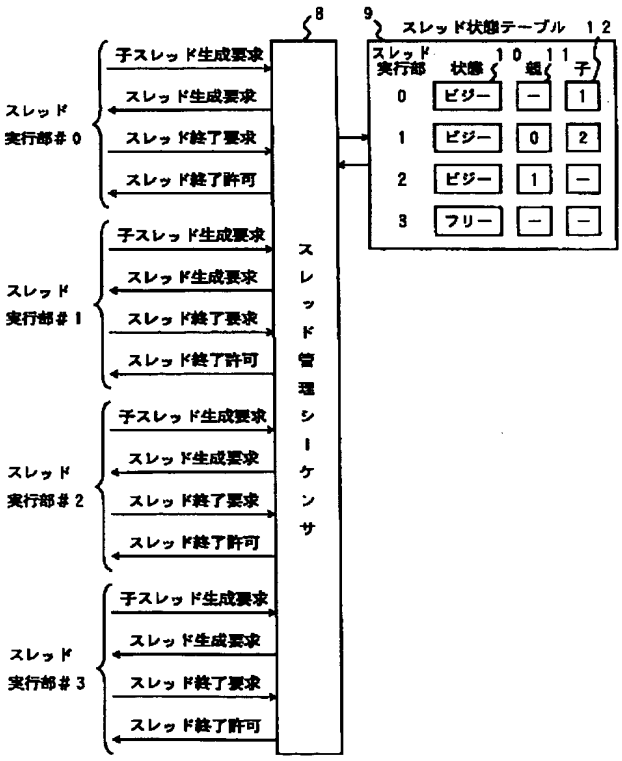
【図１６】



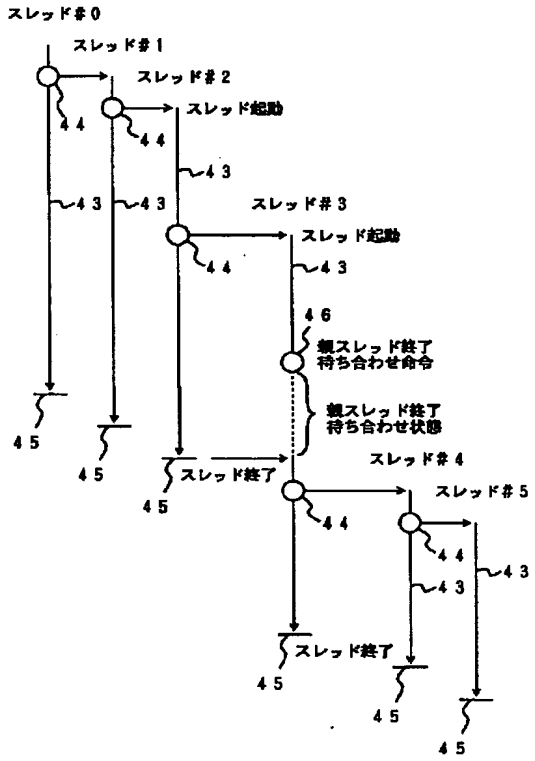
【図３】


スレッド管理部
5
スレッド実行部
＃０
スレッド実行部
＃１
スレッド実行部
＃２
スレッド実行部
＃３
6 a
6 b
6 c
6 d
7 スレッド実行部間交信バス

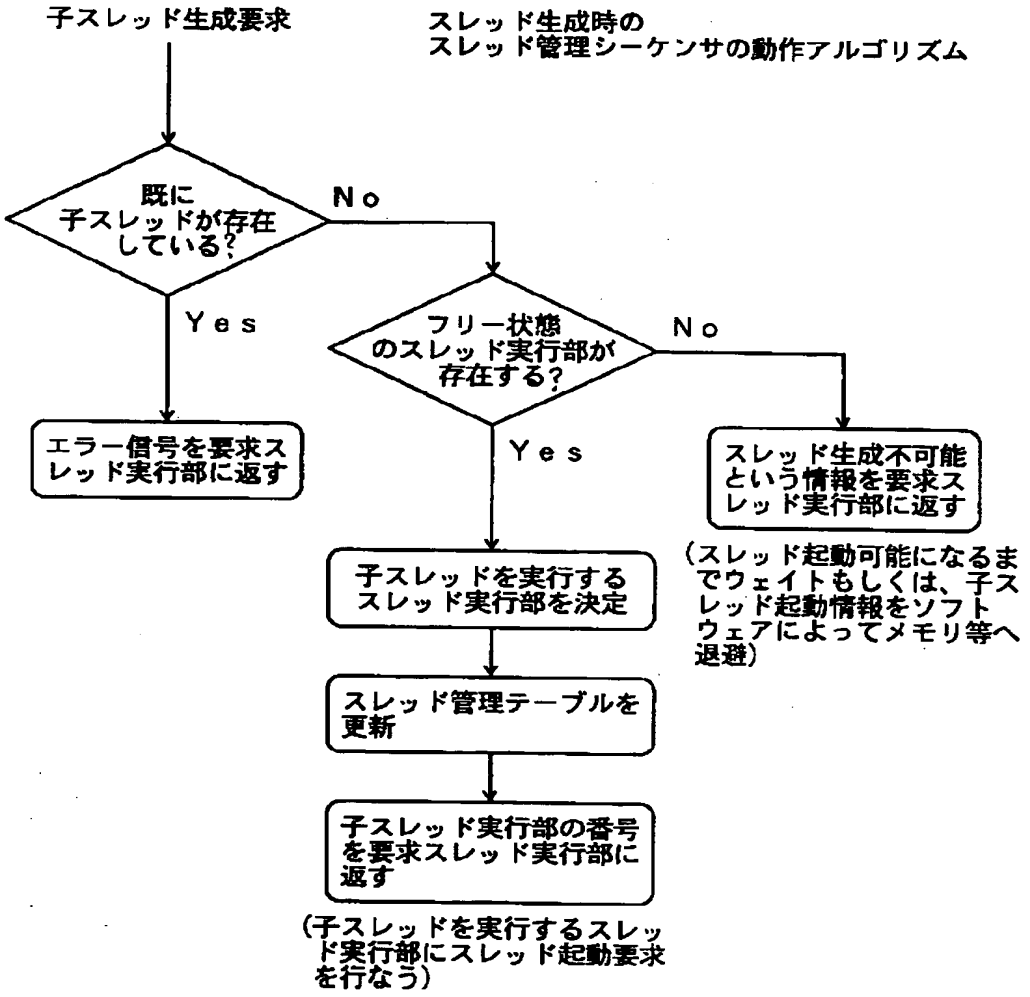
【図４】


8
9
スレッド状態テーブル
12
スレッド実行部
状態
10
親
11
子
0 ビジー − 1
1 ビジー 0 2
2 ビジー 1 −
3 フリー − −
スレッド実行部＃０
スレッド実行部＃１
スレッド実行部＃２
スレッド実行部＃３
子スレッド生成要求
スレッド生成要求
スレッド終了要求
スレッド終了許可
スレッド管理シーケンサ

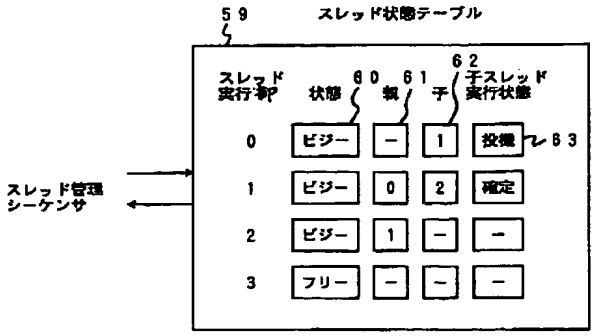
【図１７】


スレッド＃０
スレッド＃１
スレッド＃２
スレッド起動
スレッド＃３
スレッド起動
親スレッド終了
待ち合わせ命令
親スレッド終了
待ち合わせ状態
スレッド＃４
スレッド＃５
スレッド終了
スレッド終了
44
43
45
46

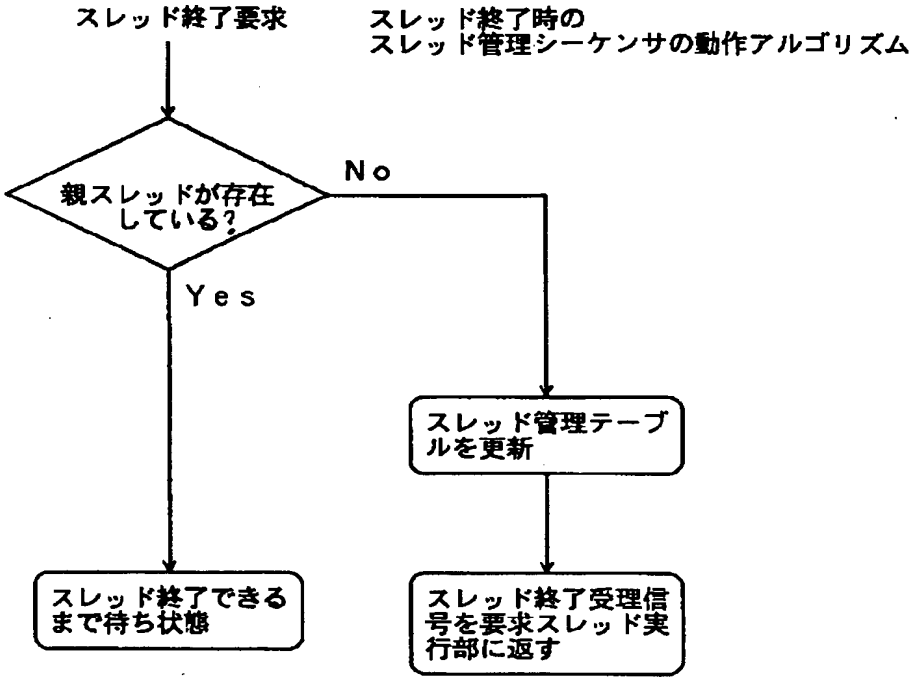
【図５】


子スレッド生成要求
スレッド生成時の
スレッド管理シーケンサの動作アルゴリズム
既に
子スレッドが存在
している?
No
Yes
フリー状態
のスレッド実行部が
存在する?
No
Yes
エラー信号を要求ス
レッド実行部に返す
スレッド生成不可能
という情報を要求ス
レッド実行部に返す
(スレッド起動可能になるま
でウェイトもしくは、子ス
レッド起動情報をソフト
ウェアによってメモリ等へ
退避)
子スレッドを実行する
スレッド実行部を決定
スレッド管理テーブルを
更新
子スレッド実行部の番号
を要求スレッド実行部に
返す
(子スレッドを実行するスレッ
ド実行部にスレッド起動要求
を行なう)

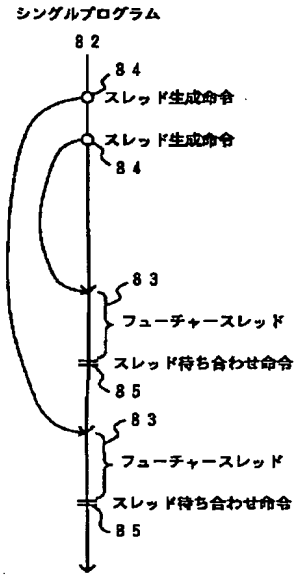
【図２１】


59
スレッド状態テーブル
スレッド
実行部P
状態
60
親
61
子
62
子スレッド
実行状態
0
ビジー
−
1
投機
63
1
ビジー
0
2
確定
2
ビジー
1
−
−
3
フリー
−
−
−
スレッド管理
シーケンサ

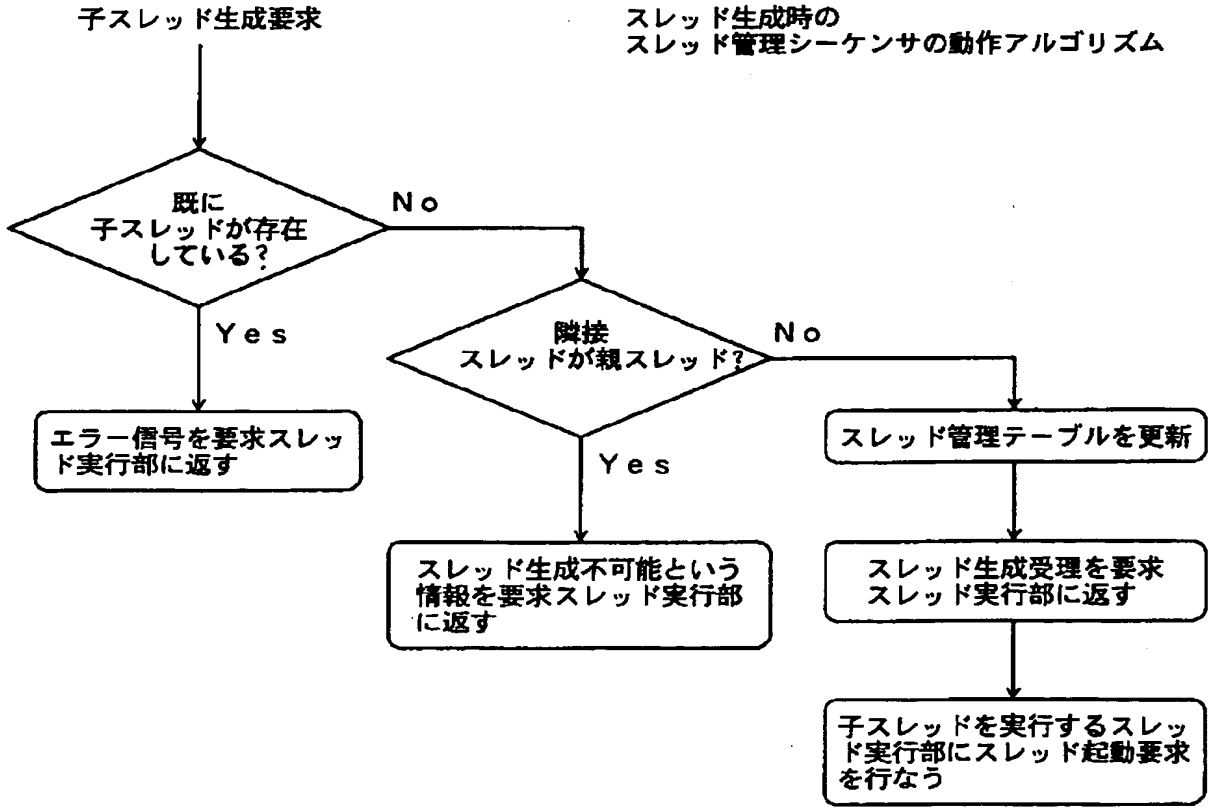
【図６】


スレッド終了要求
スレッド終了時の
スレッド管理シーケンサの動作アルゴリズム
親スレッドが存在している?
No
Yes
スレッド管理テーブルを更新
スレッド終了できるまで待ち状態
スレッド終了受理信号を要求スレッド実行部に返す

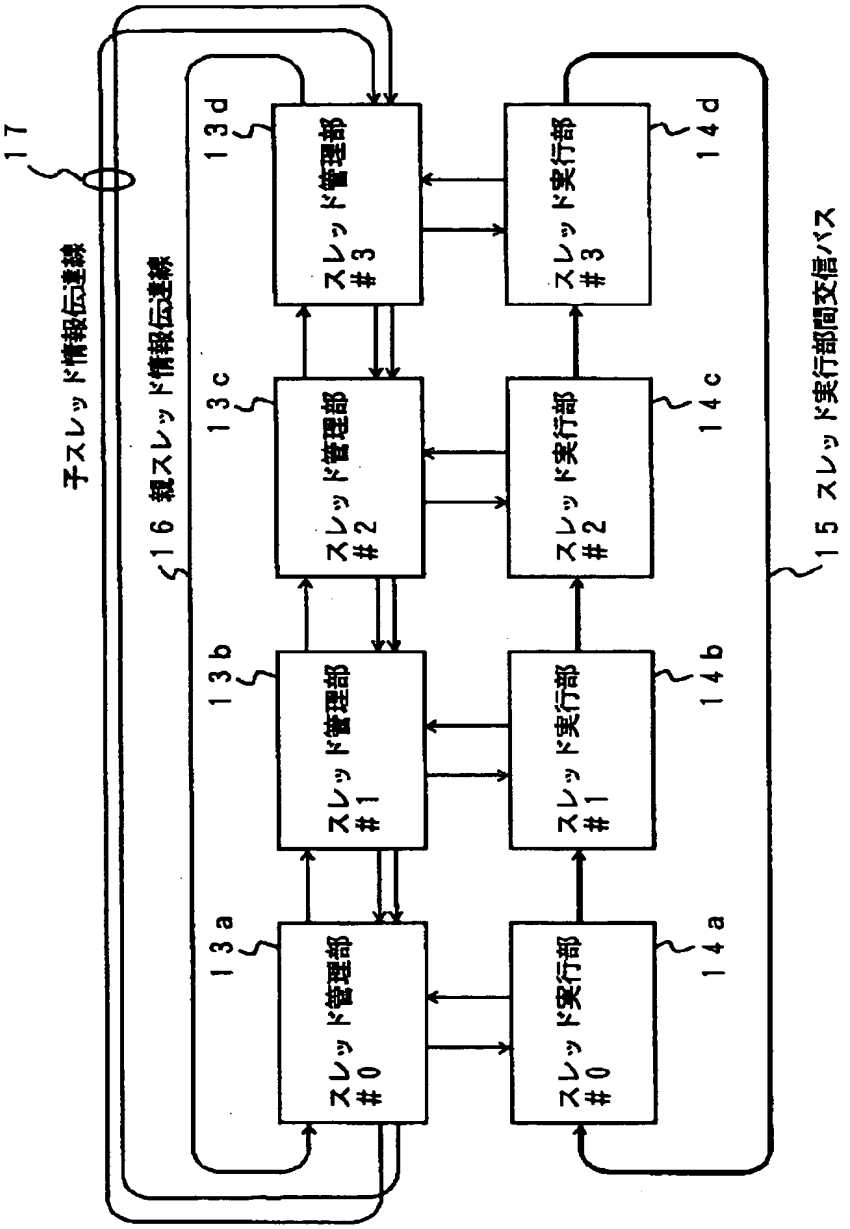
【図２４】


シングルプログラム
82
84
スレッド生成命令
スレッド生成命令
84
83
フューチャースレッド
スレッド待ち合わせ命令
85
83
フューチャースレッド
スレッド待ち合わせ命令
85

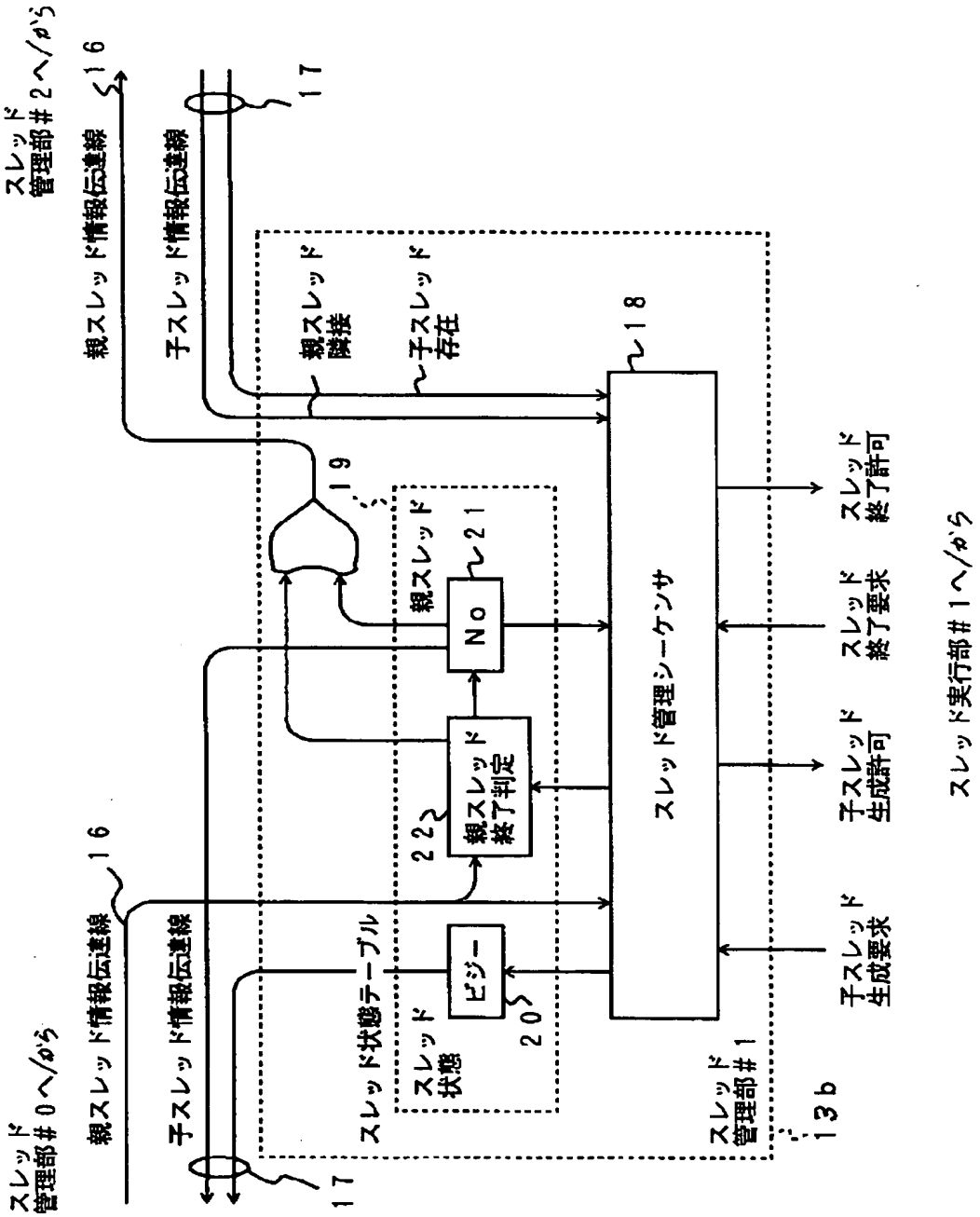
【図１０】


子スレッド生成要求
スレッド生成時の
スレッド管理シーケンサの動作アルゴリズム
既に子スレッドが存在している?
No
Yes
隣接スレッドが親スレッド?
No
Yes
エラー信号を要求スレッド実行部に返す
スレッド生成不可能という情報を要求スレッド実行部に返す
スレッド管理テーブルを更新
スレッド生成受理を要求スレッド実行部に返す
子スレッドを実行するスレッド実行部にスレッド起動要求を行なう

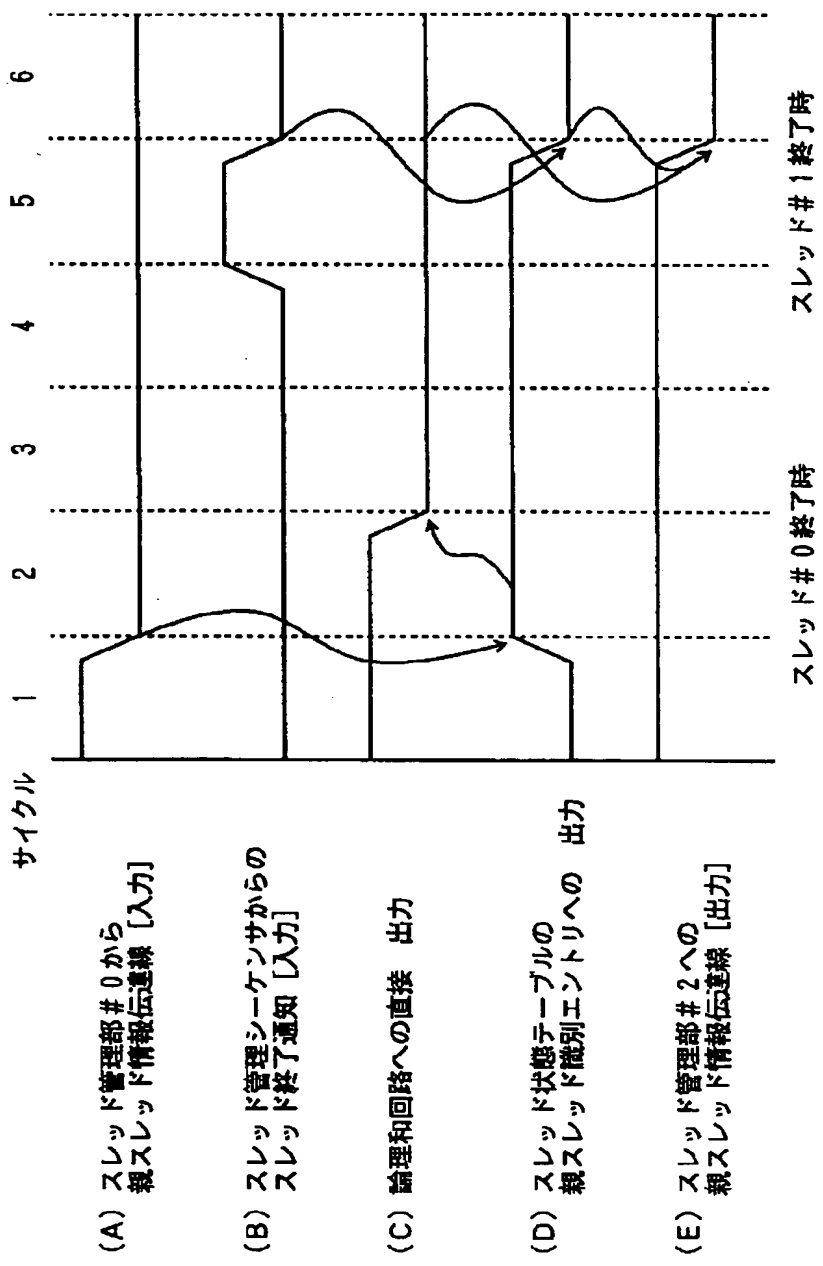
【図７】


１７
子スレッド情報伝達線
１６ 親スレッド情報伝達線
１３ｄ
スレッド管理部
＃３
１３ｃ
スレッド管理部
＃２
１３ｂ
スレッド管理部
＃１
１３ａ
スレッド管理部
＃０
１４ｄ
スレッド実行部
＃３
１４ｃ
スレッド実行部
＃２
１４ｂ
スレッド実行部
＃１
１４ａ
スレッド実行部
＃０
１５ スレッド実行部間交信バス

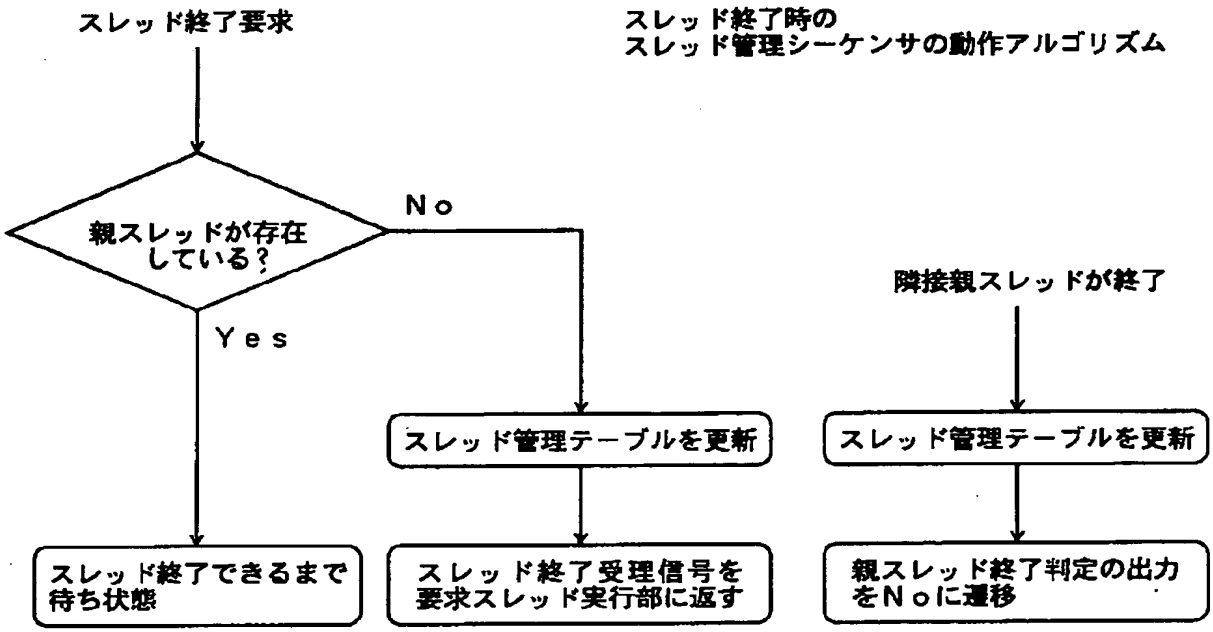
【図８】


スレッド管理部＃２へ/から
親スレッド情報伝達線
16
17
子スレッド情報伝達線
親スレッド隣接
子スレッド存在
18
19
親スレッド
21
No
22
親スレッド終了判定
スレッド状態テーブル
スレッド状態
ビジー
20
スレッド管理シーケンサ
スレッド終了許可
スレッド終了要求
子スレッド生成許可
子スレッド生成要求
スレッド管理部＃1
13b
スレッド実行部＃1へ/から
スレッド管理部＃0へ/から
親スレッド情報伝達線
16
子スレッド情報伝達線
17

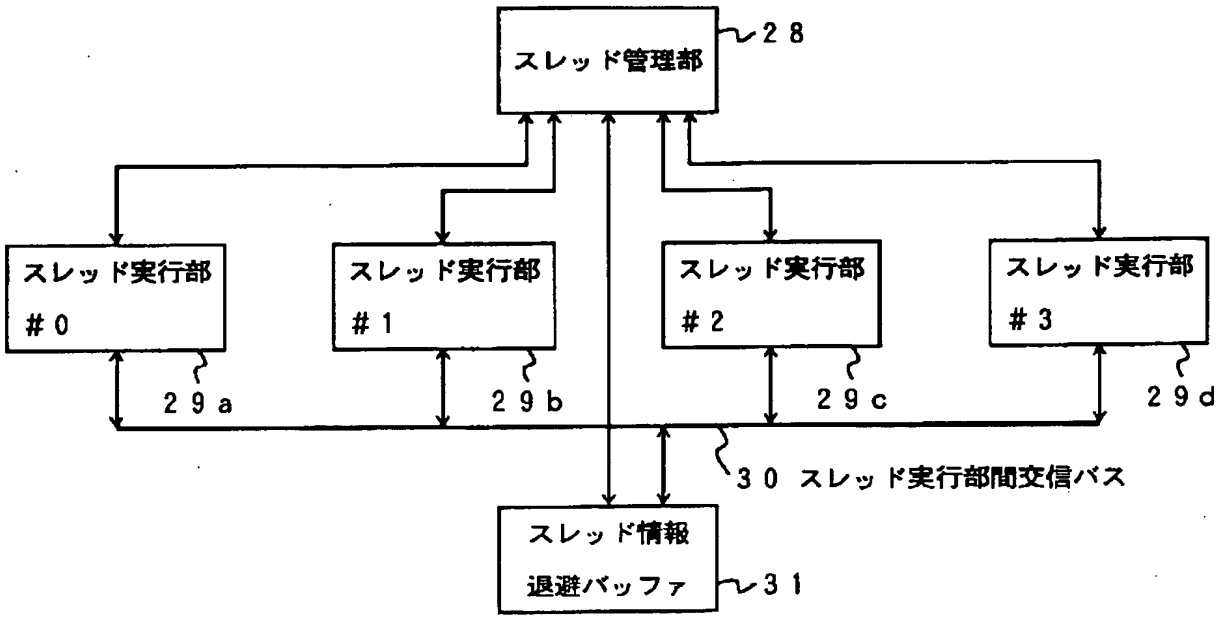
【図９】


サイクル
1
2
3
4
5
6
スレッド＃０終了時
スレッド＃１終了時
(A) スレッド管理部＃０から
親スレッド情報伝達線［入力］
(B) スレッド管理シーケンサからの
スレッド終了通知［入力］
(C) 論理和回路への直接　出力
(D) スレッド状態テーブルの
親スレッド識別エントリへの　出力
(E) スレッド管理部＃２への
親スレッド情報伝達線［出力］

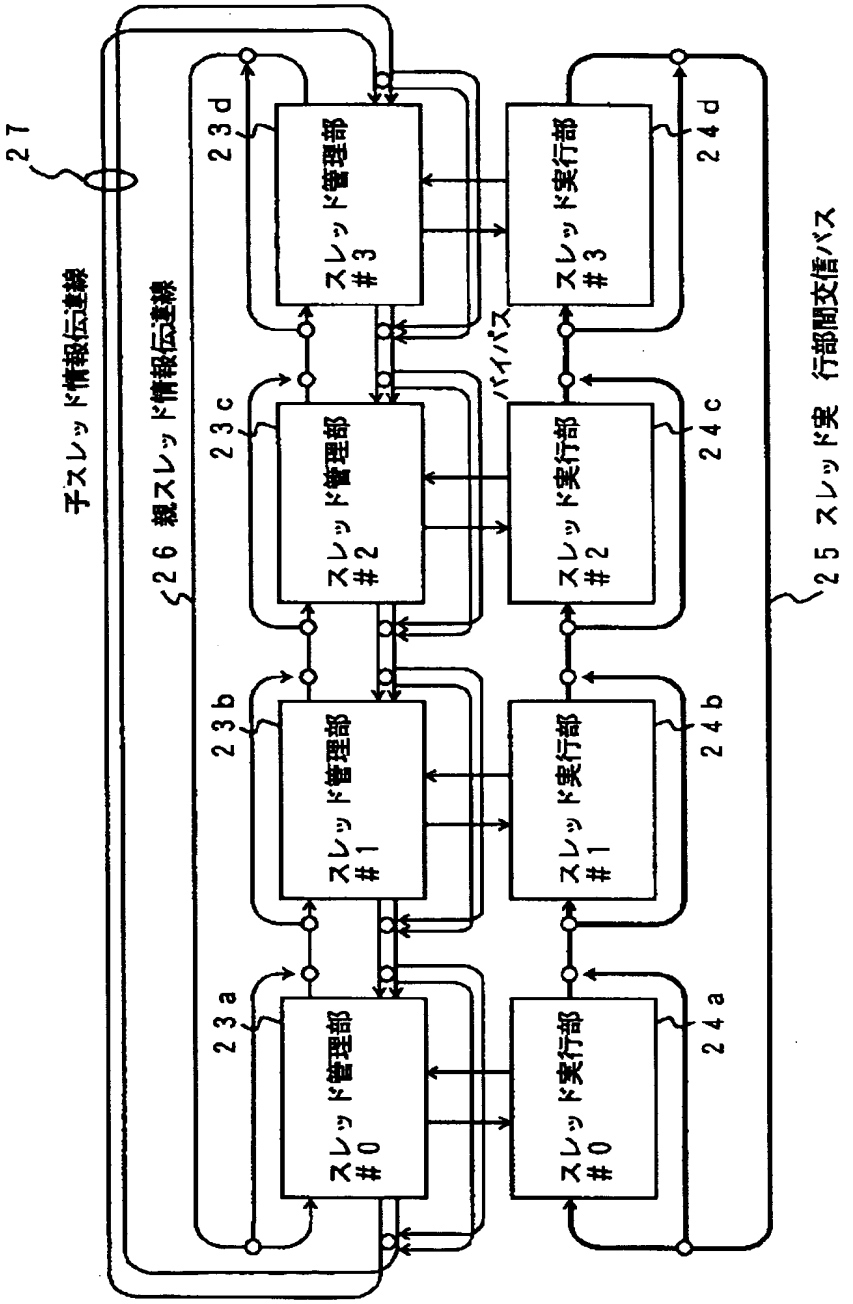
【図１１】


スレッド終了要求
スレッド終了時の
スレッド管理シーケンサの動作アルゴリズム
親スレッドが存在
している？
Ｎｏ
Ｙｅｓ
隣接親スレッドが終了
スレッド管理テーブルを更新
スレッド管理テーブルを更新
スレッド終了できるまで
待ち状態
スレッド終了受理信号を
要求スレッド実行部に返す
親スレッド終了判定の出力
をＮｏに遷移

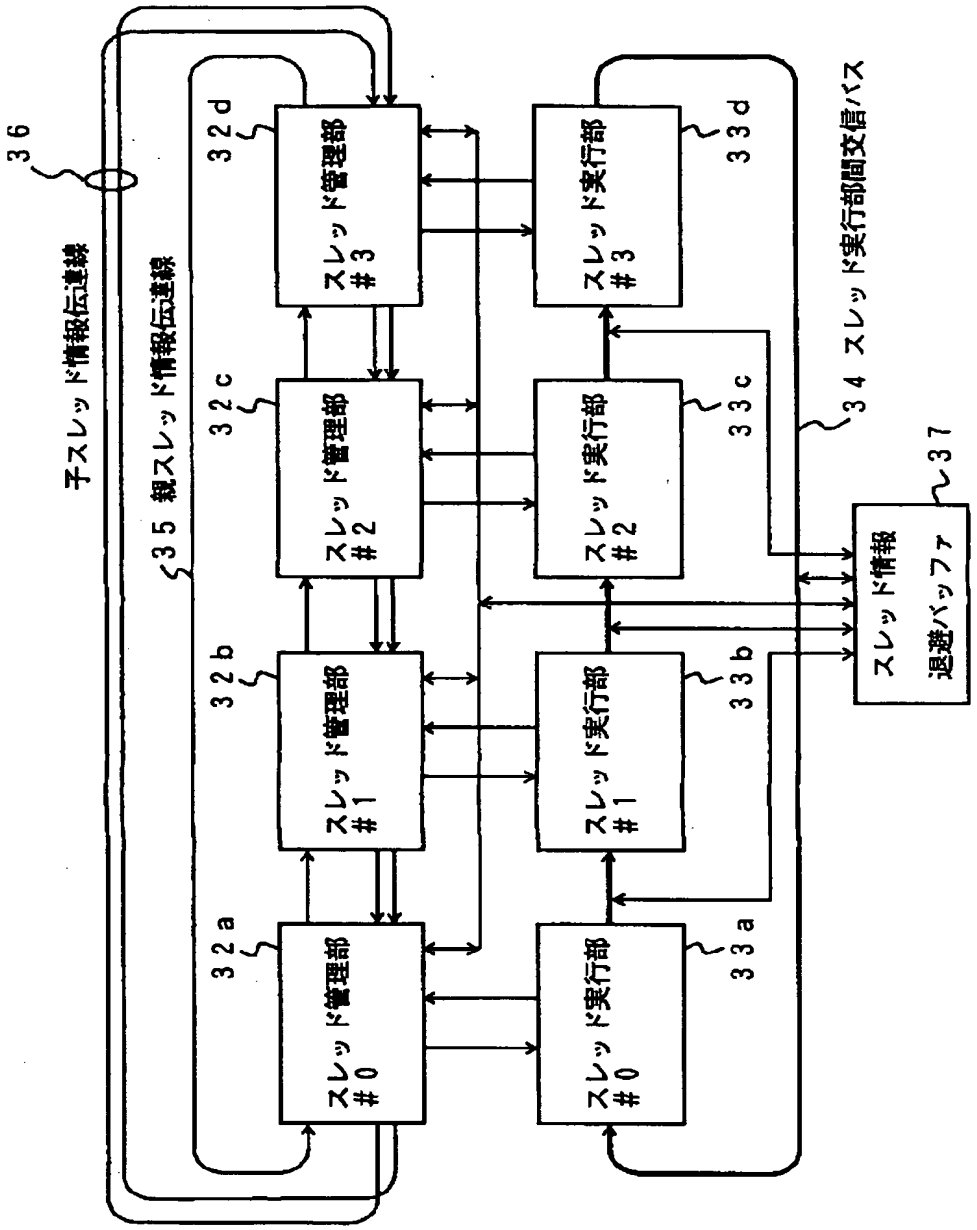
【図１３】


スレッド管理部
２８
スレッド実行部
＃０
スレッド実行部
＃１
スレッド実行部
＃２
スレッド実行部
＃３
２９ａ
２９ｂ
２９ｃ
２９ｄ
３０　スレッド実行部間交信バス
スレッド情報
退避バッファ
３１

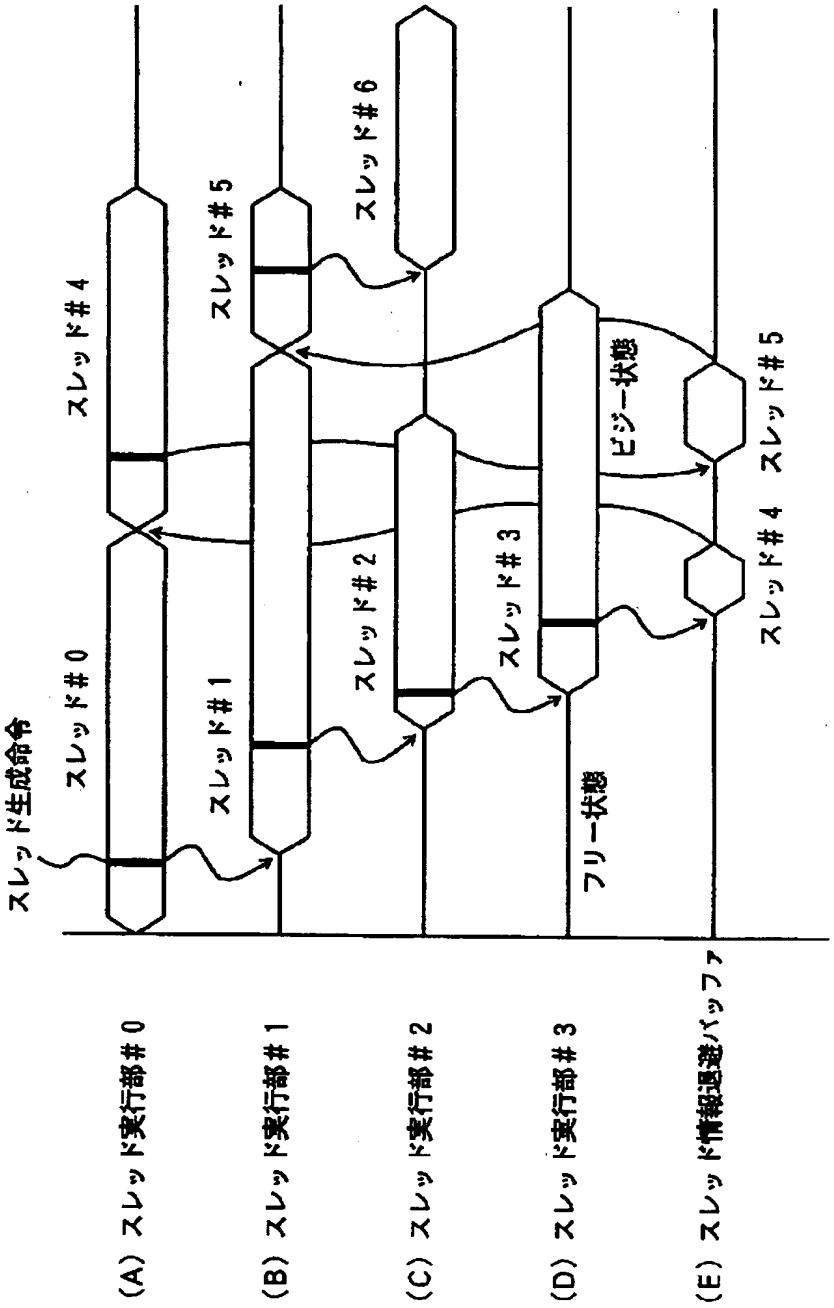
【図１２】


２７
子スレッド情報伝達線
２６ 親スレッド情報伝達線
２３ｄ
スレッド管理部 #３
２３ｃ
スレッド管理部 #２
２３ｂ
スレッド管理部 #１
２３ａ
スレッド管理部 #０
バイパス
スレッド実行部 #３
２４ｄ
スレッド実行部 #２
２４ｃ
スレッド実行部 #１
２４ｂ
スレッド実行部 #０
２４ａ
２５ スレッド実　行部間交信バス

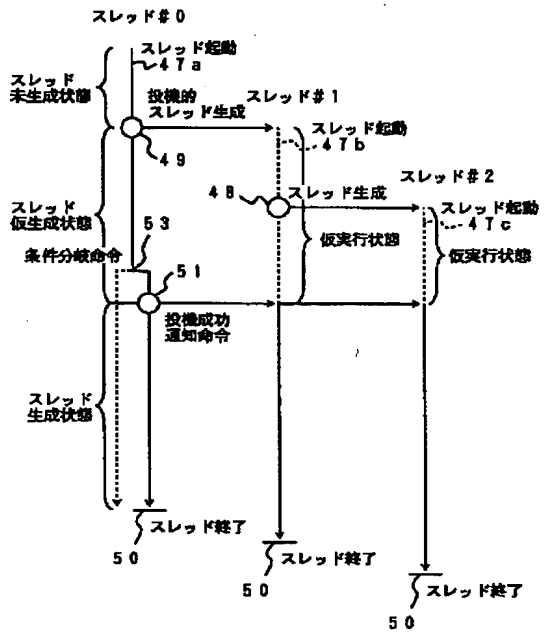
【図１４】


３６
子スレッド情報伝達線
３５ 親スレッド情報伝達線
３２ｄ
スレッド管理部 #３
３２ｃ
スレッド管理部 #２
３２ｂ
スレッド管理部 #１
３２ａ
スレッド管理部 #０
３３ｄ
スレッド実行部 #３
３３ｃ
スレッド実行部 #２
３３ｂ
スレッド実行部 #１
３３ａ
スレッド実行部 #０
３４ スレッド実行部間交信バス
３７
スレッド情報 退避バッファ

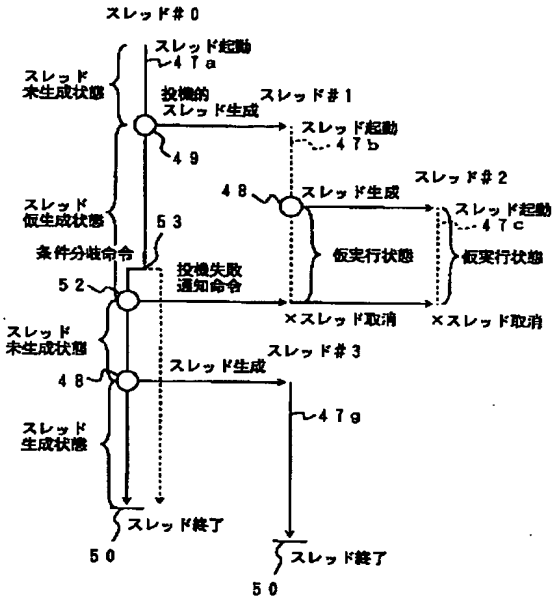
【図１５】


スレッド生成命令
スレッド＃0
スレッド＃1
スレッド＃2
スレッド＃3
スレッド＃4
スレッド＃5
スレッド＃6
フリー状態
ビジー状態
スレッド＃4
スレッド＃5
(A) スレッド実行部＃0
(B) スレッド実行部＃1
(C) スレッド実行部＃2
(D) スレッド実行部＃3
(E) スレッド情報退避バッファ

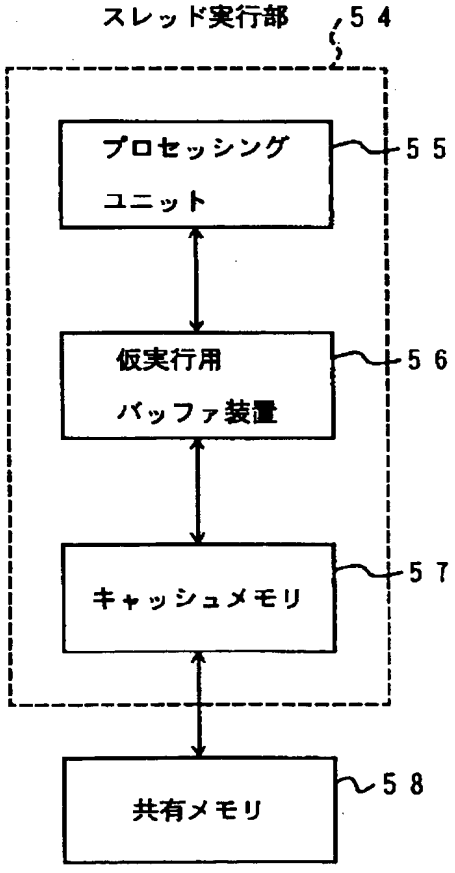
【図１８】


投機が成功する場合
スレッド＃０
スレッド起動
４７ａ
スレッド未生成状態
投機的スレッド生成
スレッド＃１
スレッド起動
４７ｂ
４９
４８
スレッド生成
スレッド＃２
スレッド起動
４７ｃ
スレッド仮生成状態
５３
条件分岐命令
５１
仮実行状態
仮実行状態
投機成功通知命令
スレッド生成状態
スレッド終了
５０
スレッド終了
５０
スレッド終了
５０

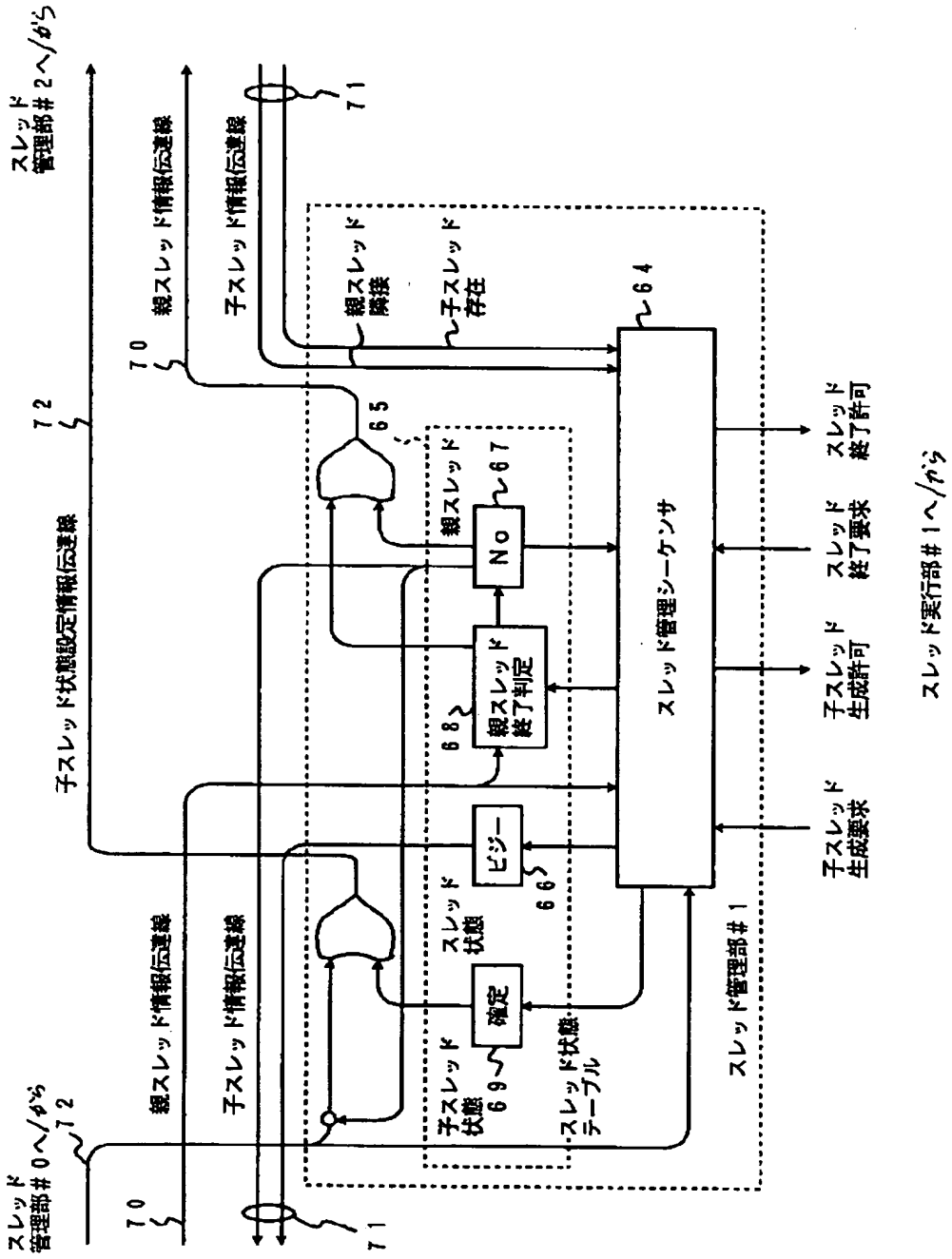
【図１９】


投機が失敗する場合
スレッド＃０
スレッド起動
４７ａ
スレッド未生成状態
投機的スレッド生成
スレッド＃１
スレッド起動
４７ｂ
４９
４８
スレッド生成
スレッド＃２
スレッド起動
４７ｃ
スレッド仮生成状態
５３
条件分岐命令
仮実行状態
仮実行状態
５２
投機失敗通知命令
×スレッド取消
×スレッド取消
スレッド未生成状態
スレッド＃３
４８
スレッド生成
４７ｇ
スレッド生成状態
スレッド終了
５０
スレッド終了
５０

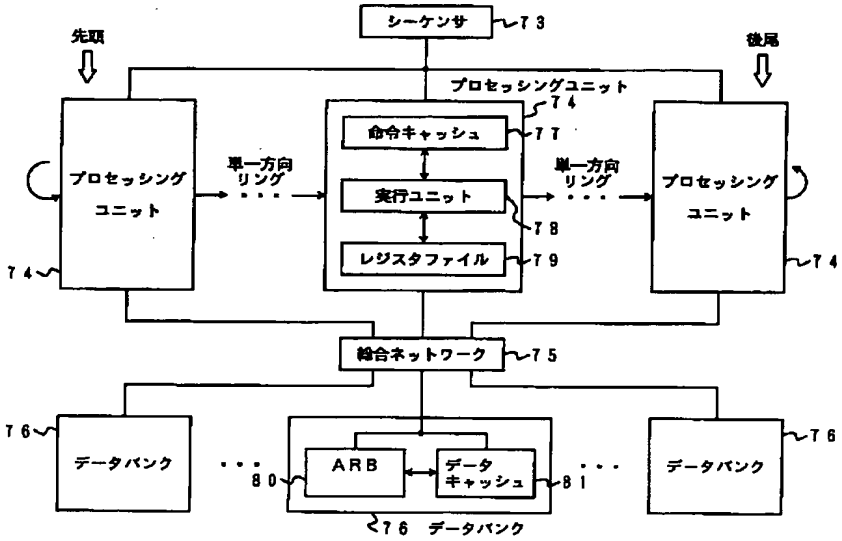
【図２０】


スレッド実行部
５４
プロセッシングユニット
５５
仮実行用バッファ装置
５６
キャッシュメモリ
５７
共有メモリ
５８

【図２２】


スレッド管理部＃２へ/から
スレッド管理部＃０へ/から
スレッド管理部＃１
スレッド実行部＃１へ/から
親スレッド情報伝達線
子スレッド情報伝達線
子スレッド状態観定情報伝達線
親スレッド隣接
子スレッド存在
スレッド管理シーケンサ
スレッド状態テーブル
子スレッド状態
スレッド状態
親スレッド終了判定
No
ビジー
確定
子スレッド生成要求
子スレッド生成許可
スレッド終了要求
スレッド終了許可
70
71
72
64
65
66
67
68
69

【図２３】


シーケンサ
73
先頭
後尾
プロセッシングユニット
74
命令キャッシュ
77
実行ユニット
78
レジスタファイル
79
単一方向リング
単一方向リング
プロセッシングユニット
プロセッシングユニット
74
74
総合ネットワーク
75
76
データバンク
80
ARB
データキャッシュ
81
76
データバンク
76 データバンク